Simulation réaliste de ruisseaux en temps réel

Frank Rochet

Université Joseph Fourier de Grenoble Master 2 IVR 2005

Simulation réaliste de ruisseaux en temps réel

Frank ROCHET

Rapport de projet de M2R

Evasion/GRAVIR/IMAG-INRIA. UMR CNRS C5527.

Composition du jury :

Fabrice	NEYRET	Directeur de stage
Olivier	BERTRAND	Rapporteur externe
Augustin	LUX	Examinateur
Nicolas	HOLZSCHUCH	Examinateur
James	CROWLEY	Examinateur

Table des matières

Chapitre 1

Introduction

1.1 Cadre de travail

Ce stage a été effectué au sein de l'équipe EVASION qui fait partie du laboratoire GRAVIR, dans les locaux de l'INRIA rhône-alpes à Montbonnot(38).

Le laboratoire GRAVIR (GRAphique, VIsion et Robotique) est dédié à la recherche dans le domaine de la synthèse d'images, de la vision par ordinateur et de la robotique. GRAVIR a pour tutelles le CNRS, l'INPG, l'INRIA et l'UJF. En outre, GRAVIR fait parti de l'IMAG, qui regroupe les laboratoires suivants : CLIPS, GRAVIR, ID, LEIBNIZ, LMC, LSR, TIMC et VERIMAG.

Le projet EVASION (Environnements Virtuels pour l'Animation et la Synthèse d'Images d'Objets Naturels) regroupe six chercheurs ou enseignants-chercheurs permanents, onze étudiants en thèse et deux ingénieurs experts. Ses travaux de recherche sont dédiés à la synthèse de scènes naturelles, animées ou statiques.

1.2 Problématique

De nombreux phénomènes naturels composent nos paysages et parmi eux, les étendues d'eau tels que les lacs, les mers, et les rivières font partie des plus complexes et des plus fascinants. L'un des nombreux défis de l'informatique graphique actuelle est de représenter ces phénomènes animés efficacement avec un rendu le plus réaliste possible. Avec la montée en puissance des machines actuelles et leur cartes graphiques, il devient possible de traiter des scènes naturelles complexes, par exemple au cinéma en post-processing ou dans les jeux vidéos. Cette approche des phénomènes naturels est différente de celle utilisée par les physiciens ou les numériciens, qui essayeront de modéliser le phénomène le plus fidèlement possible, alors que le graphiste voudra avoir le phénomène le plus réaliste possible au niveau visuel, peu importe la méthode utilisée.

Néanmoins, il est fréquent de voir des méthodes qui ont recours à la physique et aux méthodes de résolution numérique pour simuler des phénomènes naturels, ce qui permets de profiter des avancées de ces autres disciplines en matière de simulation. Dans le cas des fluides, cette démarche comporte de nombreux inconvénients :

- Les outils issus de la mécanique des fluides (comme la résolution Eulerienne de Navier-Stokes) sont particulièrement coûteux, d'autant plus que les scènes naturelles requièrent souvent à la fois des domaines larges et une résolution fine.
- Dans le cas de l'eau, on ne voit souvent que l'interface du fluide et du milieu environnant, et les indices secondaires du mouvement (vagues, ondes de surface, objets dérivant). Il est évident que dans le cas de la synthèse d'image on veut éviter de simuler avec précision ce qui est invisible.
- Ces indices secondaires sont des phénomènes émergeant des équations de Navier-Stokes dans son cas le plus général (non-linéaire, compressible, grandes vitesses, etc). Ils sont donc délicats et coûteux à obtenir.
- Pour la même raison, bien que ces phénomènes détiennent le sens de ce que l'on perçoit de l'écoulement, il est extrêmement difficile de contrôler directement leurs caractéristiques, alors que l'une des propriétés essentielles d'une technique de synthèse d'images est sa contrôlabilité par l'utilisateur.

Ces remarques nous ont conduit à baser notre approche sur une méthode phénoménologique, qui simule uniquement les éléments visibles du fluide. Cette approche est compatible avec la contrainte de rendu temps réel fixée pour ce mémoire, notre objectif sera donc de proposer une modélisation et un rendu efficace de phénomènes observables dans un ruisseau. Nous allons pour cela utiliser une variété de techniques de modélisation et d'affichage disponible dans la littérature, que nous allons exposer dans un premier temps, avant d'expliquer le fonctionnement de la méthode qui nous avons développée pour le rendu. Nous terminerons par un compte-rendu des résultats obtenus, qui offrent de nombreuses perspectives d'amélioration et de perfectionnement pour les années à venir.

Chapitre 2

Contexte du problème

2.1 Contexte de travail et terminologie

2.1.1 Contexte de travail

Notre approche de rendu de ruisseaux en temps réel s'insère dans la continuité d'un projet plus vaste qui a pour objectif la simulation et le rendu photo-réaliste de ruisseaux en temps réel. Ainsi, les méthodes de rendu présentées ici s'appuient sur une simulation de ruisseaux effectuée lors du stage de Natalie Praizelin, présentée dans [NP01]. Cette simulation s'intéresse principalement aux ondes de choc crées par les obstacles émergeant de l'eau. Ces ondes de choc sont construites géométriquement et sont donc accessible sous forme vectorielle, afin d'être affichées. C'est à partir des ces lignes et bandes que notre approche va construire une représentation du phénomène visuel observable, plus ou moins détaillée selon le besoin. Dans les zones calmes où aucune onde n'est visible, nous représenterons la surface de l'eau de façon très efficace. Dans les zone de vague, la surface sera représentée par le strict nécessaire.

Comme nous l'avons dit, nous allons nous concentrer sur le rendu de ruisseaux en temps réel. Nous allons donc reproduire les attributs pertinents de la surface du ruisseau ainsi que les effets optiques qui s'y produisent. Un ruisseau implique une eau peu profonde, un mouvement quasi-stationnaire, globalement calme. Les phénomènes pertinents observables sur de telles étendues d'eau sont les ondes de choc en amont et en aval des obstacles, les ondes capillaires de moindre intensité qui précédent les ondes de choc, et les remous perturbateurs, c'est donc essentiellement sur ces phénomènes macroscopiques que nous nous concentrerons. Comme nous avons pour contrainte de faire l'animation en temps réel, nous utiliserons le hardware programmable des cartes graphiques pour gérer des phénomènes optiques.

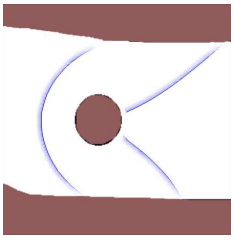


FIG. 2.1 – Un onde de choc tirée de [NP01] (le courant va de gauche à droite)

2.1.2 Terminologie

Tout au long de cette étude nous allons souvent parler de vague, d'onde et d'autres éléments qui composent l'apparence de la surface d'un ruisseau. Il est important de fixer une terminologie à ce propos puisque les termes peuvent être ambigus et qu'il n'existe pas vraiment de "standard" en la matière. Comme illustré par la figure ci-dessous (figure 2.2), la partie calme du ruisseau (où il n'y pas de vague) est appelée le plan d'eau, puisque dans notre cas nous l'assimilerons à un plan. Dans la rivière nous avons un certain nombre d'*obstacles* qui génèrent des *vagues*, matérialisées entre autres par leur *axe principal* et leur *épaisseur* qui définissent le *support* de la vague. La vague peut être vue comme une extrusion du *profile* le long du support, le profile étant la caractérisation même de la forme de vague, contenant *l'onde de choc* et des *ondes capillaires*.

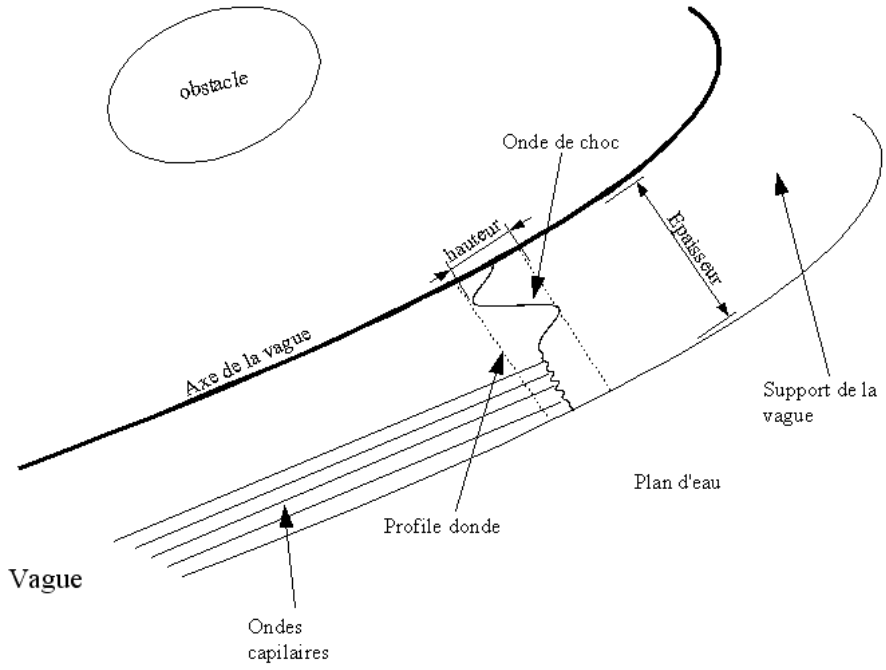


FIG. 2.2 – Terminologie utilisée

2.2 Simulation de fluides

De nombreuses méthodes de simulation des fluides sont utilisées en animation graphique. On peut les regrouper en trois grands domaines.

2.2.1 Modèles physiques (CFD)

Simulations Euleriennees (Grilles)

Les simulations basées la résolution numérique des modèles de la mécanique des fluides (Navier-Stokes) considèrent la totalité du fluide et le simulent avec précision. Une discrétisation est nécessaire pour que ces modèles soient applicables à l'informatique graphique, puisqu'ils reposent sur la résolution numérique d'équations différentielles, qui permettent de simuler de la fumée et des fluides 3D turbulents [Sta99, Lov03]. La précision de la simulation dépend essentiellement de la finesse de la

discrétisation, et pour obtenir des phénomènes pertinents comme des ondes de choc ou des ondes capillaires il faut utiliser une grille de résolution très fine, ce qui a pour effet d'augmenter considérablement les temps de calcul.

Simulations Lagrangiennes (Particules)

Une autre représentation d'un liquide en informatique graphique peut être obtenue grâce à un système de particules. Dans ce cas, la finesse de la représentation sera en fonction de la taille des particules et de leur nombre. Ce modèle permets de simuler des phénomènes difficiles à obtenir avec d'autres approches comme des chutes d'eau, des cascades, des geysers ou des fontaines. Il a l'avantage de n'imposer aucune contrainte spatiale au liquide, mais il est néanmoins nécessaire d'avoir un très grand nombre de particules pour obtenir un résultat réaliste, ce qui réserve cette méthode aux petites étendues d'eau ou aux phénomènes turbulents, et aux rendus non temps-réel.

Modèles spécifiques

D'autres approches s'attaquent à des modèles plus spécifiques (donc plus simples) déduits de Navier-Stokes, comme la propagation des ondes de surfaces pour simuler l'apparence de la mer [FR86].

2.2.2 Modèles spectraux et procéduraux

Les approches spectrales produisent un champ de hauteur obéissant au spectre de la surface de l'eau, sans se préoccuper de sa structure interne (motifs). Cela peut être fait en filtrant un bruit blanc par un spectre ayant les propriétés requises et en calculant la transformée de Fourier rapide (FFT) du résultat. Ces approches [Sta01, Lov03] ont tendance à perdre des phénomènes locaux ou persistants comme les remous, et ne produisent pas forcément une animation de bonne qualité malgré leur pertinence statistique.
Certains autres modèles simplifiés à l'extrême utilisent des sommes de sinus plutôt qu'un spectre, combinées empiriquement pour obtenir une simulation très simple à coder. On les rencontre souvent dans les jeux vidéos [Fin04].

2.2.3 Modèles phénoménologiques

Les méthodes phénoménologiques simulent directement le comportement du fluide (ici l'apparence de la surface) sans se préoccuper de leurs causes microscopiques et privilégient généralement le qualitatif à l'exactitude numérique. Ce genre d'approche fait appel à des modèles simples basés sur l'observation des manifestations en surface du fluide, la connaissance de son mouvement et sa physique à l'échelle macroscopique. En 1802, Gerstner propose un modèle de la houle d'un océan avec des trochoïdes, qui sert de base à de nombreuses simulations océanographiques ou graphiques. Le modèle proposé par [NP01] a été spécialement développé pour la simulation de ruisseaux, basé sur l'observation des ondes de choc produites en amont ou en aval des obstacles immergés dans un ruisseau. C'est le modèle sur lequel nous allons nous baser pour notre travail de construction et de rendu de surface d'eau, pour lequel il nous fournit la description animée des ondes.

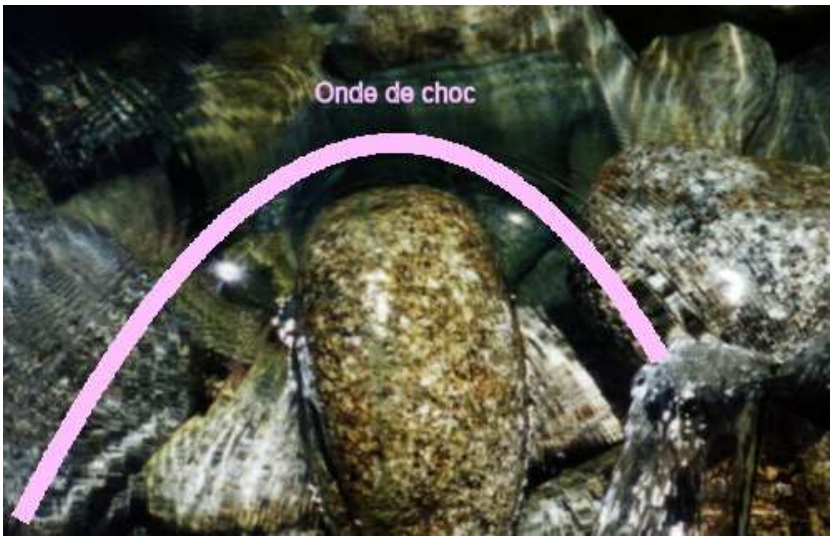


FIG. 2.3 – Photographie d'un système d'ondes typique au voisinage d'un obstacle sur un ruisseau réel. Le courant s'écoule de haut en bas. On a fait figurer l'onde de choc. Noter les ondes capillaires (millimétriques) en amont.

Pour la surface d'un ruisseau peu profond, la situation est similaire à l'air, le régime supersonique ou subsonique caractérisé par le nombre de Mach $M = \frac{Vair}{Vondessonores}$ se transposant en nombre de Froude $Fr = \frac{Veau}{Vondesdesurface}$. Les ondes de choc produites par ces obstacles, où s'accumulent les ondes de Froude, peuvent être construites géométriquement à partir du point le plus haut sur la courbe de l'iso-Froude, où le nombre de Froude Fr = 1. En général, les ruisseaux sont super-critiques (Fr > 1, en bleu clair sur la figure 2.4), mais dans les zones proches des obstacles, il arrive qu'ils soient sous-critique (Fr < 1, en bleu foncé sur la figure 2.4).

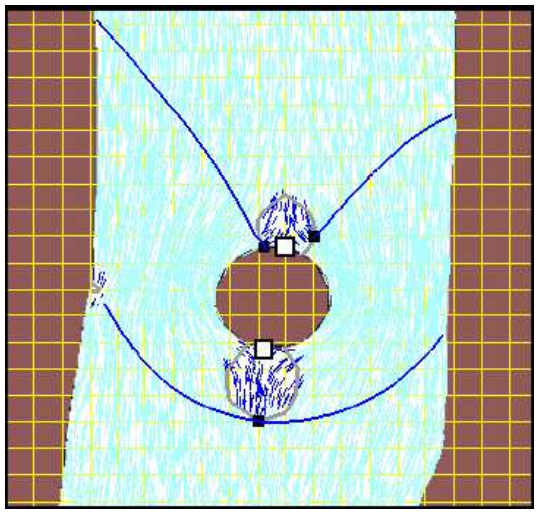


FIG. 2.4 – Zones sur et sous-critiques autour de l'obstacle, et les ondes de choc associées (le courant vas de bas en haut)

Dans un fluide aux vitesses homogènes, les ondes de Froude (ondes de choc microscopiques) vues de dessus sont des cônes 2D, d'angle d'ouverture $\arcsin(\frac{1}{Fr})$, comme un cône de bang supersonique. Si les vitesses ne sont pas homogènes, les courbes représentant l'onde de Froude s'incurvent localement. Si l'on admet que le fluide est incompressible et irrotationnel, on peut estimer les vitesses aux sommets d'une grille grâce à une équation de Laplace ($\nabla\phi = 0, v = \nabla\phi$), puis interpoler les valeurs (voir grille et vitesse interpolées sur la figure 2.4). On construit ensuite les ondes de choc avec la méthode expliquée précédemment, ce qui nous donne un construction vectorielle virtuellement indépendante de la résolution de sortie. L'écoulement du ruisseau étant supposé quasi-stationnaire, ces vitesses sont calculées une fois pour toutes, puis perturbées par des flotteurs aléatoires. Dans l'implémentation utilisée dans [NP01], la forme et la profondeur du ruisseau et des obstacles est donnée par une image en niveaux de gris.

2.3 État de l'art

Mon travail consiste en la visualisation réaliste d'une surface d'eau en temps réel. Dans cette section je passe donc en revue les travaux et concepts existants dans 3 disciplines : la représentation géométrique de surfaces détaillées, le rendu (optique physique), et l'utilisation du hardware graphique (GPU). Concernant l'animation, qui est en amont de mon sujet, on se référera à la section 2.2

2.3.1 Représentation géométrique de la surface

Si l'on fait abstraction des effets optiques qui caractérisent l'eau, et que l'on se concentre sur la modélisation de sa surface, on peut se dire qu'il suffit de choisir une grille à très grosse résolution pour notre maillage de la surface et qu'ainsi on obtiendra une surface d'eau la plus détaillée possible. Cette approche simpliste est loin d'être optimale : la fabrication des triangles, leur transfert à la carte graphique et leur rendu prendraient un temps prohibitif incompatible avec le rendu temps réel, d'autant plus que le ruisseau existe aussi en dehors du champ visuel. De plus, dans le cas de fluides calmes comme une rivière, il y a beaucoup d'endroits sur la surface de l'eau où rien ne se passe et où l'on a pas besoin de résolution. Si la grille n'est pas assez fine (il faut un pas d'ordre du millimètre pour représenter les ondes capillaires), il est possible de rater des détails ou de subir des artefacts en dents de scie ou des pointillés à cause du manque de résolution, on aura donc de l'*aliasing*.

MIP-Mapping

Le MIP-Mapping est une forme d'anti-aliasing courrament utilisée de nos jours. On l'utilise pour éviter l'aliasing sur des textures détaillées, qui se matérialise sous la forme de figures de Moiré caractéristiques. Quand la texture est sur un polygone lointain, il y a plus de texels (pixel de la texture) que de pixels à rendre sur le polygone, certains texels sont sautés, et on perd de l'information visuelle. Pour obtenir un anti-aliasing mathématiquement correct, il faudrait utiliser tous les texels qui contribuent à chaque pixel sur l'écran, ce qui est difficile en temps réel puisqu'un seul pixel peut correspondre à des centaines de texels. L'idée du MIP-mapping [EWWL98] est d'utiliser une grosse texture quand on dessine un gros polygone, et de diminuer sa taille à mesure que le polygone deviens plus petit à l'écran. Les versions plus petites de la texture (les *"MIP-maps"*) sont calculées en faisant la moyenne pondérée d'une version plus grande. Si on rend un polygone avec une texture qui a environ le même nombre de texels que ce polygone a de pixel à l'écran, les figures de Moiré disparaissent.

2.3.2 Niveaux de détails

Nous avons besoin de détail à l'avant plan, mais si on modélise tout le ruisseau à cette résolution, le coût deviens prohibitif. Pire : dans le lointain, l'accumulation de détails dans un même pixel provoque une autre forme d'aliasing, toute aussi désagréable à l'oeil. Il devient donc rapidement évident qu'il est inefficace de recourir à un seul à un unique maillage constant, indépendant du point de vue et de la distance. Les standard de niveaux de détails continus (CLOD) comme ROAM [DWS+97] ou le morphing de quad-tree [CE01] sont trop coûteux et rarement adaptés à des surfaces en mouvement (ces techniques ont été développées pour la modélisation de terrains). Dans ce cas, on a recours soit à la multi-résolution, soit à une hiérarchie de modèles, soit à une combinaison des deux. Une hiérarchie de modèles [Kaj85] consiste à remplacer les détails géométriques par autre chose quand ils sont trop loin pour être discernés, en général on utilise le bump mapping (cf. ci-dessous) puis la réflectance (i.e. un matériau dont la rugosité représente les détails), mais des approches plus approximatives à base de texture peuvent être envisagées. Notre méthode propose une approche composée d'une hiérarchie de modèles multi-résolution spécifiquement développée pour le rendu de vagues à base de géométrie et de bump mapping. Un point important est d'assurer entre les différentes résolutions et les différents modèles : la transition doit s'opérer en douceur et l'oeil humain ne doit pas la percevoir [Bec92].

Bump mapping Il est donc possible de rendre l'aspect de petits détails en utilisant une surface plane et une carte de normales pour perturber les normales à la surface de l'eau de façon à donner une impression de relief si on la regarde avec un angle pas trop rasant. Cette technique, appelée Bump Mapping, est très largement utilisée pour le rendu de l'eau puisque qu'elle donne de très bons résultats visuels et qu'elle est très rapide [AVO02, Bel03].

Textures animées Perlin a utilisé des approche de synthèse de bruit [Per85] pour simuler l'apparence de la surface de l'océan vue depuis une distance lointaine. On peut utiliser cette technique pour simuler la surface de l'eau en temps réel avec une texture animée, ce qui permets un affichage temps réel très performant, mais qui manque tout de même de réalisme, la surface étant plane, ce qui constitue une grossière approximation de la surface de l'eau, restant toutefois valable pour les points de vue lointains et non rasants. Une approche spécifiquement adaptée au mouvement des fluides à été proposée dans [PN01].

Modélisation de surface de fluides

La surface d'une rivière est une forme complexe, et plusieurs techniques différentes sont actuellement utilisées pour modéliser efficacement cette surface, selon les contraintes et les besoins. Comme nous l'avons vu précédemment, dans le cas d'un rendu temps réel, ces techniques seront combinées selon leurs caractéristiques (performance, qualité visuelle) pour obtenir le résultat le plus réaliste possible.

Génération de maillage 3D La solution la plus évidente lorsque l'on a une simulation de fluide basée sur une grille de résolution fixe est de mailler directement la surface à partir d'un champ de hauteurs issu de la simulation. Cette solution procure un rendu satisfaisant si on maille suffisamment finement la surface, ce qui deviens vite très coûteux. Dans une solution temps réel, elle sera en général utilisée uniquement pour les vagues ayant une grande longueur d'onde, les ondes de moindre importance comme les capillaires seront traitées avec une autre technique (par exemple en bump mapping comme dans [Lov03]).

Grille déformable par GPU Une approche très actuelle pour le rendu d'eau en temps réel consiste à utiliser un maillage statique et planaire, modifié par un champ de hauteurs résultant de la simulation. Les sommets du maillage seront déformés en temps réel par le processeur graphique selon les variations de la surface de l'eau dictées par la simulation, qui elle aussi peut être calculée par le processeur graphique. L'énorme avantage de cette technique est sa rapidité, puisqu'il s'agit d'une méthode temps réel par essence. Ses inconvénients principaux sont le manque de souplesse et la résolution statique, la simulation devant être effectuée par le processeur graphique, actuellement limité en flexibilité de calcul et en nombre d'instructions (cf. section sur le hardware programmable). De nombreux démos de fluides temps réel utilisent cette technique (dans [Zel04] entre autres), mais elles représentent des surfaces liquides très petites.

Systèmes de particules Si la simulation utilisée est basée sur un système de particules, on dispose d'un panel de méthodes de la plus simple (et la plus rapide) à la plus complexe pour en effectuer le rendu : on peut utiliser de simples points (qui auront en général une taille variable), des sprites pour donner aux particules l'aspect de gouttes d'eau par exemple, ou encore générer des surfaces implicites comme dans [EMF02], ce qui permet d'obtenir un rendu photo-réaliste de l'eau en mouvement avec des interactions complexes.

2.3.3 Optique physique

Lorsque nous allons effectuer le rendu de la géométrie issue de la simulation, nous allons utiliser les lois de l'optique physique pour simuler le comportement de la lumière à travers l'étendue d'eau. Les lois qui régissent ces effets sont connues depuis plusieurs centaines d'années, nous allons présenter ici les plus fondamentales pour notre étude : la réflection, la réfraction, et le coefficient de Fresnel.

Réflection

Si un rayon de lumière se propage dans un milieu homogène et arrive à la surface d'un second milieu homogène, une partie de la lumière est réfléchie et l'autre partie pénètre dans le second milieu avec une distortion (réfraction) et peut ou non y être absorbé.

La proportion de la lumière réfléchie dépend du rapport des indices de réfraction des deux milieux considérés. Le plan d'incidence est défini par le rayon incident et la perpendiculaire, ou normale, à la surface au point d'incidence du rayon (voir figure 2.5). L'angle d'incidence est l'angle formé par le rayon incident et cette normale. Les angles de réflexion et de réfraction sont les angles entre la normale et les rayons réfléchis et réfractés. D'après les lois de la réflexion de Snell-Descartes, l'angle de réflexion est égal à l'angle d'incidence.

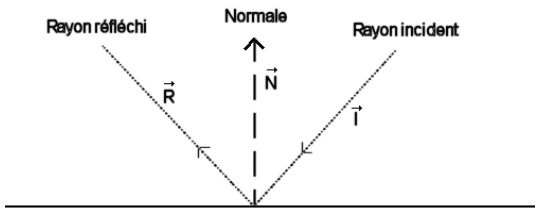


FIG. 2.5 – Lois fondamentales de la réflexion

On peut calculer le rayon réfléchi grâce à la formule :

$$\vec{R} = \vec{I} - 2\vec{N}(\vec{N} \cdot \vec{I})$$

Si la surface du second milieu est lisse, il peut agir comme un miroir et produire une image réfléchie (voir figure 2.6). On considère qu'un point d'une source lumineuse A émet des rayons lumineux dans toutes les directions. Les deux rayons qui parviennent au miroir aux points B et C sont réfléchis. La réflexion produit respectivement les rayons (BD) et (CE). Pour un observateur situé devant le miroir, ces rayons semblent provenir d'un point unique F placé derrière le miroir. D'après les lois de la réflexion, les rayons (CF) et (AC) forment le même angle avec la surface du miroir. Il en est de même pour (AB) et (BF). Ainsi, dans le cas d'un miroir plan, l'image d'un objet semble être formée par les rayons issus d'une source située derrière le miroir à une distance de la surface du miroir égale à la distance entre la surface et l'image. Nous aurons exactement les même propriétés optiques si A représente l'oeil d'un observateur et DE le paysage vu en réflection dans le miroir BC.

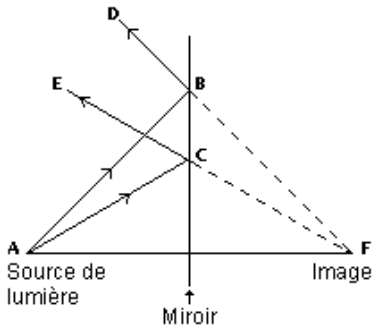


FIG. 2.6 – Réflexion sur un miroir plan

Si la surface du second milieu est rugueuse, les normales microscopiques aux différents points de la surface ont des directions différentes, que l'on peut considérer aléatoires si le milieu est suffisamment rugueux. Dans ce cas, les rayons émis d'un point source sont si dispersés qu'ils ne peuvent former une image.

Réfraction

Le rayon réfracté produit par une surface transparente ou semi-transparente obéit à la loi de Snell-Descartes :

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

$$\frac{\sin\theta_t}{\sin\theta_i} = \frac{\eta_i}{\eta_t} = \eta_r$$

η_i est l'indice de réfraction du matériau duquel provient le rayon lumineux, et η_t est l'indice de réfraction du matériau dans lequel il va pénétrer.

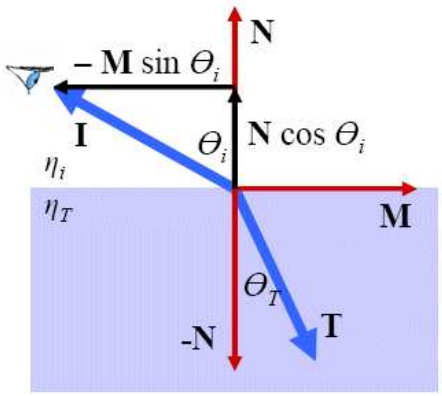


FIG. 2.7 – Réfraction : Loi de Snell-Descartes

Ici, $\vec{I}$ représente le rayon lumineux incident, $\vec{N}$ la normale à la surface transparente, et le vecteur $\vec{M}$ le projeté orthogonal du rayon incident dans le plan de réflection et $\vec{T}$ le rayon réfracté. On a donc :

$\vec{I} = \vec{N}\cos\theta_i - \vec{M}\sin\theta_i$
$\vec{M} = (\vec{N}\cos\theta_i - \vec{I})/\sin\theta_i$
$\vec{T} = -\vec{N}\cos\theta_t + \vec{M}\sin\theta_t$

D'où l'on tire la formule courament utilisée pour la réfraction :
$\vec{T} = (\eta_r(\vec{N}\cdot\vec{I}) - \sqrt{1-\eta_r^2(1-(\vec{N}\cdot\vec{I})^2)})\vec{N} - \eta_r\vec{I}$

Le rayon incident, le rayon réfracté et la normale au point d'incidence sont dans le même plan. En général, l'indice de réfraction d'un matériau transparent dense est plus élevé que pour un matériau transparent moins dense (exemple : air = 1, eau = 1.33, diamant = 2.4 [Wlo04]), cela signifie que plus un milieu est dense, plus la vitesse de la lumière s'y propageant est faible. Ainsi, un rayon se propageant dans un milieu et qui pénètre dans un second milieu d'indice de réfraction supérieur se rapproche de la normale au plan d'incidence, alors qu'un rayon pénétrant dans un milieu d'indice de réfraction inférieur s'en éloignera. Un exemple classique illustre ce phénomène : le décalage de la position lors de l'observation d'un objet sous l'eau par une personne située au-dessus de la surface de l'eau (figure 2.8).

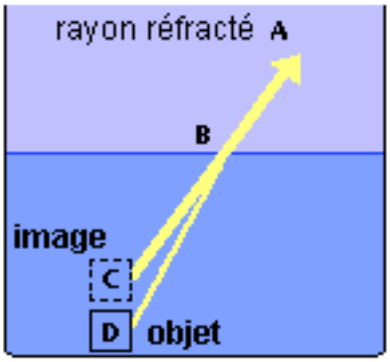


FIG. 2.8 – Un objet placé dans l'eau observé depuis l'extérieur

Le rayon (DB) provenant de l'objet en D s'écarte de la perpendiculaire vers A. L'objet, de ce fait, semble être situé en C où la ligne ABC est sécante d'une droite perpendiculaire à la surface de l'eau et passant par D. la réfraction fait donc apparaître les objets immergés plus proches de la surface de l'eau qu'ils ne le sont réellement.

Fresnel

Comme nous venons de le voir, lorsque un rayon de lumière se propage dans un milieu homogène et arrive à la surface d'un second milieu homogène, une partie de la lumière est réfléchie et l'autre partie pénètre dans le second milieu (réfraction) et peut ou non y être absorbé. Le coefficient qui détermine la proportion de lumière qui est réfléchie et la proportion qui est réfractée est appelée coefficient de Fresnel $R(\theta)$, et il est calculé comme suit (Nb : on utilise l'otique simplifiée sans polarisation) :

$$c = \cos(\theta_i)\eta_r$$

$$g = \sqrt{1 + c^2 - \eta_r^2}$$

$$R(\theta) = \frac{1}{2}((g-c)/(g+c))^2(1+[(c(g+c)-\eta_r^2)/(c(g-c)+\eta_r^2)^2)]$$

2.3.4 Hardware graphique programmable

Notre méthode fait une utilisation intensive du hardware graphique actuellement disponible afin de pouvoir s'effectuer en temps réel. Il y a quelques années, les cartes graphiques permettaient uniquement de traiter la partie "rasterisation" du pipeline graphique, c'est dire de dessiner des triangles à l'écran. Les calculs de transformation et d'illumination étaient à la charge du processeur central. Au fil des années, les constructeurs de hardware graphique ont transféré la totalité du pipeline dans la carte graphique. Les calculs graphiques étant par nature facilement parallélisables, un énorme gain de performance à pu être observé ces dernières années grâce à des solution matérielles toujours plus spécialisées, allant même jusqu'à dépasser la loi de Moore. On parle donc maintenant de GPU, pour "Graphic Processing Unit".

Shaders programmables

Les deux parties principales du pipeline graphique, le traitement des sommets et le dessin des triangles sont longtemps restées figées, le développeur pouvant utiliser telle ou telle fonctionnalité présente sur la carte graphique. Initialement, le pipeline standard, aujourd'hui souvent appelé "pipeline fixe", transformait les sommets dans le repère de la caméra, les projetait à l'écran, faisait les calcul d'illumination à chacun de ces sommets, puis rasterisait les triangles en interpolant les couleurs à chacun de ces sommets, avec éventuellement des possibilités de texturing. Aujourd'hui, deux parties du pipeline graphique sont programmables, via des langages dédiés : la transformation des sommets, et le calcul de la couleur des pixels, ceci permet une grande liberté de traitement de l'information, ce qui eu pour effet de faire exploser les possibilités d'effets visuels sur GPU. On parle de "Vertex Shader" ou de "Vertex program" pour les programmes de traitement des sommets et de "Pixel Shader" ou "Fragment program" pour les programmes de traitement au niveau pixel [JR03].

Stencil Buffer

Le stencil buffer permets de créer des masques d'écriture qui détermineront si chaque pixel est écrit ou non dans le frame buffer. C'est particulièrement utile pour créer des ombres (stencil shadows) ou résoudre des problèmes de visibilité, ou tout autre problème d'affichage complexe qui nécessite un traitement par pixel. Le fonctionnement du stencil est décrit en détail dans le "red book" OpenGL [NDW93].

Environment mapping

Dans le cas de calculs de réflections en temps réel, il est bien sur impensable d'aller intersecter chaque rayon de lumière réfléchi avec la géométrie complexe qui compose le monde environnant de l'objet dans le cas d'un rendu temps réel. On utilise donc souvent une approximation qui consiste à encoder ce qui entoure l'objet (l'environnement) dans une texture (l'environment map), puis ensuite d'aller chercher les informations de réflection dans cette texture en fonction du rayon de lumière réfléchi. Ce procédé à été développée en 1976 par Blinn et Newell [BN76].

Bump mapping

En 1978, James Blinn introduit la notion de Bump Mapping [Bli78], qui donne l'illusion de profondeur sur des surfaces apparemment lisses, grâce à des effets de relief. On calcule une carte de normales (*normal map*) à partir d'une carte de hauteur, qui est ensuite utilisée pour représenter la normale à chaque pixel du polygone à rendre, et qui va enfin servir à des calculs d'illumination par pixel.

2.3.5 Effets optiques sur GPU

Le pipeline programmable de la carte graphique graphique va nous permettre de programmer un certain nombre d'effets visuels indispensables à l'obtention d'un ruisseau réaliste. Les lois optiques physiques que nous avons évoquées sont pour certaines applicables tel quelles. D'autres, pour obtenir des effets temps réels, doivent passer par des simplifications et des approximations plus ou moins poussées. En général un panel de techniques, de l'approximation la plus grossière à la reproduction quasi-fidèle sont disponibles, parmi lequel choisira le meilleur compromis entre la qualité de rendu et le temps d'exécution.

Réflection

Dans la figure 2.5, on peut utiliser une camera située au point F pour simuler les réflections locales sur l'objet A, en effectuant un rendu dans une texture, ce qui signifie que l'on stocke dans une texture ce qui est visible depuis ce point de vue. Ensuite on *reprojette* cette texture sur l'objet A, c'est à dire la texture agit comme une diapositive placée devant un projecteur dirigé vers la surface réfléchissante. On obtiens avec cette méthode une bonne approximation des réflections sur cet objet. Dans le cas du rendu d'une étendue d'eau, on effectue donc un rendu en deux passes : on calcule la position de la camera miroir par rapport à la surface réfléchissante (ex : un plan d'eau), on effectue un rendu dans une texture, puis on projette cette texture sur la surface réfléchissante dans une deuxième passe. Ce type d'environment mapping fait appel aux textures projectives [SKvW+92]. De nombreux articles sur le rendu de l'eau sur GPU utilisent cette technique [Lom04, Bel03, Zel04, AVO02].

Une autre alternative consiste à supposer que l'observateur est infiniment lointain, auquel cas on peut considérer que les rayons incidents sont sont tous parallèles, et donc les rayons réfléchis ne dépendent que de la normale à la surface (et plus de la distance). Si l'on suppose également que les objets composant l'environnement sont infiniment lointains, ce qui est souvent le cas lorsque l'on représente des paysages naturels en extérieur, la carte d'environnement peut être encodée sous la forme d'un cube sur lequel chacune des faces est plaquée une texture qui représente ce qui est visible depuis l'objet (appelée *Cube Map*). Cette technique ne permet pas de représenter correctement les réflections locales puisque l'environnement est supposé infiniment lointain, elle est néanmoins couramment utilisée pour le rendu d'étendues d'eau [Lom04, Bel03]. C'est cette technique qui à été retenue dans notre implémentation de la réflection pour sa simplicité de mise en place.

Réfraction

Les mêmes techniques temps réel que pour la réflection pourront être adoptées pour la réfraction : environment mapping, textures projectives. En ce qui concerne la méthode avec texture projective, elle est décrite en détail dans [Bel03] et dans [AVO02]. On remplace la camera de réflection par une camera de réfraction (qui est la même que la camera principale), et on effectue un déplacement empirique guidé par la géométrie de la surface.

Fresnel

La formule donnée dans 2.3.3 est difficilement utilisable tel quel dans le cadre d'un rendu temps réel, à cause du trop grand nombre de paramètres à calculer. On utilise en général l'approximation de Schlick, qui donne de bons résultats, et qui est facilement implémentable sur le hardware graphique :

$$R_0 = \frac{1-\eta_r^2}{1+\eta_r^2}$$

$$R(\theta) = R_0 + (1-R_0)(1-cos\theta)^5$$

Parrallax mapping

L'effet de parallaxe peut être observé lorsque des portions d'une surface bougent relativement les unes par rapport aux autres quand le point de vue change. On peut observer ce phénomène sur n'importe quelle surface qui n'est pas complètement plate. Or un polygone rendu en bump mapping,

vu avec un angle assez rasant, manque d'effet de parallaxe. En effet, il aplatit la géométrie sur le polygone, ce qui enlève l'information de hauteur qui aurait été générée par un équivalent géométrique. Pour palier ce défaut, une technique nommée parrallax mapping [Wel04] (ou encore offset bump mapping ou virtual displacement mapping) à vu le jour, qui permets de tenir compte de l'information de hauteur pour corriger le manque de parallaxe, de façon très simple et efficace. D'autres techniques, qui vont de la simple rectification de coordonnées de texture en fonction de la parallaxe au pseudo lancer de rayon en temps réel, ont été développées pour simuler l'effet de parallaxe de façon encore plus réaliste.

Chapitre 3

Notre Méthode

3.1 Principe

Comme nous l'avons vu jusqu'ici, la méthode que nous allons présenter ici s'intéresse principalement aux ondes de choc provoquées par des obstacles immergés dans un ruisseau. Nous nous basons sur des bandelettes produites par le simulateur existant [NP01] le long des ondes. Pour donner chair à ces bandelettes, nous avons choisi une approche adaptative qui va permettre d'avoir un affichage temps réel avec le maximum de précision quel que soit le point de vue adopté pour regarder le ruisseau. Les ondes de choc pourront donc avoir deux représentations différentes : elles pourront soit être représentées avec de la géométrie lorsque le point de vue le nécessite (premier plan, ou vue rasante avec risque d'occultation), et ce en multi-résolution, soit être affichées avec une technique de bump mapping dont nous détaillerons le fonctionnement plus loin. Ces différentes méthodes d'affichages devront être correctement mixées avec des transitions automatiques et si possible sans transition brusque (*pop-up*) afin d'obtenir un affichage précis et fluide.

3.1.1 Choix d'un profil d'onde

Nous générons une vague en extrudant un profil 1D le long de la courbe donnée par le simulateur à chaque pas de temps. Pour choisir un profil de vague le plus proche possible de la réalité, nous nous somme appuyés sur l'observation de ruisseaux réels, sur des photos d'ondes et des études préalables comme [FM98]. Le profil que nous avons choisi (figure 3.1) comporte un creux avant l'onde de choc, puis viennent les ondes capillaires. Il est entièrement contrôlable : on peut faire varier son intensité globale, sa longueur d'onde capillaire, son intensité capillaire et son amortissement capillaire. Notons que le profil doit toujours commencer avec une hauteur et une dérivée nulles et finir de même pour se raccorder parfaitement au plan d'eau. Le profil que nous utilisons pour nos calculs est normalisé sur les deux dimensions (x et y), il sera mis à l'échelle en temps voulu lorsque les calculs le nécessiteront. Il est stocké dans un tableau à une dimension avec une résolution maximum, que l'on pourra éventuellement sous-échantillonner lors des calculs de géométrie. Notre profil à été de vague à été généré grâce à la formule suivante :

$$x \in [-3..3],\ f(x) =$$
$$2(x+\sqrt{.5})\exp(-(x+\sqrt{.5})^2) + .03H(x)(exp(-\tfrac{x}{1.5}) * \cos(\tfrac{2\Pi x}{.25}) - 1 - (\tfrac{x}{3})(\exp(-\tfrac{3}{1.5})\cos(\tfrac{6\Pi}{.25}) - 1))$$

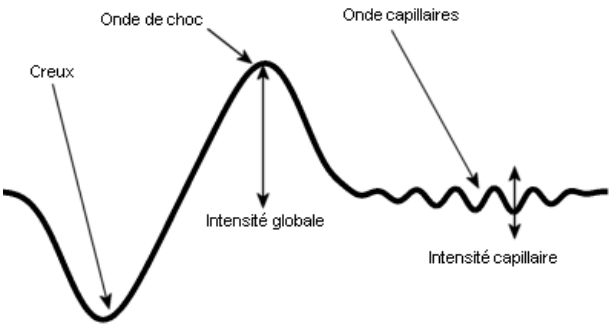


FIG. 3.1 – Le profil d'onde que nous avons choisi

3.1.2 Calcul de la visibilité des vagues

La méthode présentée ici doit permettre d'afficher un ruisseau long de plusieurs kilomètres sans que les temps de calcul soient catastrophiques. Pour ne construire que le strict nécessaire, nous devons déterminer quelles vagues sont visibles, c'est à dire présentes dans le volume de vision (*Viewing Frustum*) de la caméra. Pour cela nous allons calculer les boîtes englobantes (alignées sur les axes) des vagues en temps réel : elles vont être recalculées à chaque rendu puisque ces vagues sont dynamiques. Si la boîte englobante n'est pas contenue dans le volume de vision, la vague n'est pas visible (une méthode optimisée pour faire ce calcul est exposée en détail dans [AM00]). La méthode standard pour calculer la boite englobante d'un modèle 3D consiste à tester si chaque sommet du modèle est contenu dans la boite courante : si c'est le cas on passe au point suivant, et dans le cas contraire on l'agrandit de façon à ce qu'elle contienne le point. Il est bien sûr trop coûteux de parcourir tous les points du maillage 3D pour chaque vague, nous effectuons donc un sous-échantillonnage des points qui composent la vague, en prenant comme hypothèse la faible courbure des vagues dans le ruisseau (lors de l'affichage, nous ne traitons séparément chaque demi-ondes de choc issue du point le plus amont, ce qui permet d'admettre cette hypothèse). De plus, notre construction extrudée des vagues nous permets de calculer la boite en 2D puis de lui attribuer une hauteur pour la troisième dimension plutôt que de faire les calculs directement en 3D, ce qui nous fait gagner du temps de calcul. Dans nos expérimentations, nous n'avons pris que 5 points le long de chaque vague, répartis uniformément. Aucune erreur de boite englobante n'a été constatée avec cette méthode, dans la mesure où aucune vague ne dépasse de sa boite dans nos tests.

3.1.3 Plan d'eau

Les ondes de choc sont loin d'occuper la totalité de la surface de l'eau (figure 3.2, tirée de [NP01]). Nous allons donc rendre séparément les bandelettes des vagues, puis le plan d'eau, globalement calme et plat.

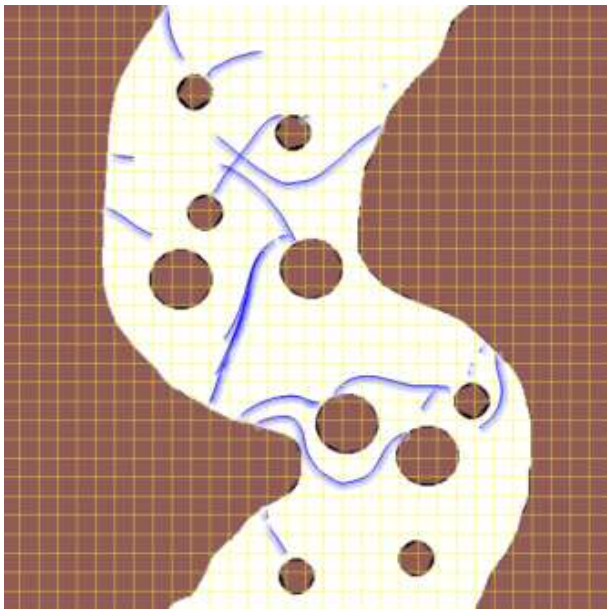


FIG. 3.2 – Les ondes de choc dans un ruisseau

En effet, nous pouvons afficher le plan d'eau sous la forme d'un seul quadrilatère (ce qui va nous permettre de l'afficher très rapidement), sur lequel nous appliquerons les effets visuels (réflexion, réfraction, Fresnel...) que nous souhaitons avoir sur le ruisseau. Les calculs pourront être simplifiés puisque la normale au plan est constante ($\vec{N} = \begin{vmatrix} 0 & 0 & 1 \end{vmatrix}$ dans notre cas). Pour chaque pixel du quadrilatère nous calculons la couleur des pixels réfléchie et réfractée, puis on mélange les deux grâce au coefficient de Fresnel, comme nous l'avons vu dans le chapitre précédent.

Si on effectue le rendu naïvement en laissant le z-buffer s'occuper de gérer la visibilité des ondes par rapport au plan, on constate que le résultat n'est pas correct. En effet, la hauteur du profil d'onde étant parfois négatif (notamment dans le creux qui précède l'onde de choc), la vague passe au dessous du plan d'eau à cet endroit et le creux n'est plus visible, ce qui est incorrect (figure 3.3).

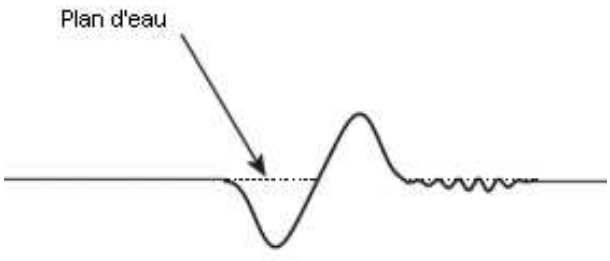


FIG. 3.3 – Le plan d'eau couvre le creux de la vague

Pour pallier ce problème, nous faisons appel au stencil buffer, avec un principe très simple :

1. On mets le stencil à zéro au début de la frame

2. On dessine les ondes en premier et on "marque" les pixels dessinés (valeur différente de zéro mise dans le stencil)
3. On dessine le plan d'eau avec le stencil test, n'autorisant le tracé uniquement là où le stencil est à zéro

3.2 Pré-calcul des normales sur le profil d'onde

Lorsque nous allons générer des primitives d'affichage pour rendre notre ruisseau (que ce soit de la géométrie ou du bump), nous allons devoir calculer les normales aux sommets du maillage ou la normal map pour le bump mapping. Nous allons pré-calculer les normales le long du profil 2D qui servira à extruder la vague, puis ces normales seront transformées à la volée pour devenir les normales finalement utilisées pour le calcul d'éclairage 3D. Plusieurs méthodes sont disponibles pour calculer les normales : nous pouvons dériver analytiquement la formule qui nous sert à générer le profil, ou bien utiliser la méthode des différences finies. Étant donnée la complexité de la formule utilisée pour modéliser le profil, nous avons préféré utiliser les différences finies, ce qui nous permet de changer plus facilement de profil et de peaufiner son aspect. Les normales du profil sont calculées comme suit :

Étant donnée $y = f(x)$ la fonction profil normalisée dans un intervalle $[0..1]$, échantillonné en S échantillons :

$$\vec{N_p}(x) = \left| \begin{array}{c} \frac{\partial_y}{\partial_x} \\ 1 \end{array} \right| = \left| \begin{array}{c} \frac{f(x-1)-f(x+1)}{2/S} \\ 1 \end{array} \right|$$

Il faut prendre quelques précautions pour que ce calcul soit valable :

- La fonction ainsi discrétisée doit être suffisamment échantillonnée, en accord avec la loi de Nyquist, c'est à dire que la fréquence d'échantillonnage doit être au moins deux fois la fréquence maximum du signal.
- Ces normales devront bien sûr être correctement transformées, mises à l'échelle et normalisées avant les calculs d'éclairage : il faut les adapter à la largeur, à l'intensité et à la courbure des bandelettes.
- Il faut veiller à ce que le profil produise des normales vers le haut en 0 et en 1 pour se raccorder parfaitement au plan d'eau. Le profil doit donc commencer et terminer à zéro avec une pente nulle.

3.3 Génération d'un maillage 3D

L'une des deux représentations retenues pour modéliser les vagues consiste donc à générer un maillage 3D à partir des informations connues sur la vague : son axe principal donné par la simulation, son profil, son épaisseur. Nous venons de voir que nous avons adopté un profil très échantillonné pour obtenir une représentation fidèle de l'onde, si nous utilisons le profil tel quel pour générer un maillage, nous obtiendrons au total des centaines de milliers de triangles, ce qui entrainera un coût d'affichage prohibitif. Nous recourons donc à un schéma multi-résolution par sous-échantillonnage du profil d'onde dans l'algorithme de génération du maillage 3D (tout en utilisant les normales pré-calculées, qui sont alors meilleures qu'une différence finie). La génération du maillage 3D elle-même

pourra donc être contrôlée avec différents niveaux de détails, conformément aux contraintes citées précédemment.

3.3.1 Génération du maillage : de la 2D à la 3D

Nous avons donc l'axe de la vague sous forme de ligne brisée 2D, que nous allons utiliser pour créer une surface maillée en 3D. Nous allons pour cela épaissir la ligne brisée en une bandelette dans le plan d'eau selon les normales pointant vers l'intérieur de la vague (que nous appellerons les directions intérieures), puis ensuite extruder le long de celle-ci le profil vertical de la vague selon la normale au plan d'eau (souvent l'axe z), et ainsi créer des quadrilatères, qui pourront éventuellement être strippés pour un affichage optimal. C'est lors de cette extrusion que nous utiliserons les facteurs de sous-échantillonnage contrôlant le niveau de détail de la vague selon l'axe et selon le profil .

Notons que si le rayon de courbure de la vague est trop fort, les directions intérieures risque de donner des quadrilatères qui se chevauchent. Pour éviter que ce phénomène de replis se produise, on effectue une relaxation sur une dizaine de directions durant leur calcul : $N'_i = \alpha N_i + (1-\alpha)N'_i - 1$, où α est le facteur de relaxation.

Nous allons maintenant expliquer en détail la création de la géométrie d'une onde et expliquer comment nous recalculons ses normales à partir de la carte de normales pré-calculée dont nous avons parlé précédemment.

Calcul des sommets du maillage

Pour générer une vague complète, nous allons parcourir la ligne brisée qui forme son axe principal à la résolution choisie, et dans chacun des quadrilatères décris par l'épaississement de cet axe le long de sa normale interne, nous allons générer un certain nombre de points, dépendants du sous-échantillonnage choisi le long de l'axe de la vague et le long de son profil (figure 3.4). Pour générer les points définissant la surface de la vague, une simple formule de changement de repère suffit :

$$\vec{P} = \vec{O} + x\vec{T} + y\vec{B} + Hf(\frac{x}{E})\vec{N}$$

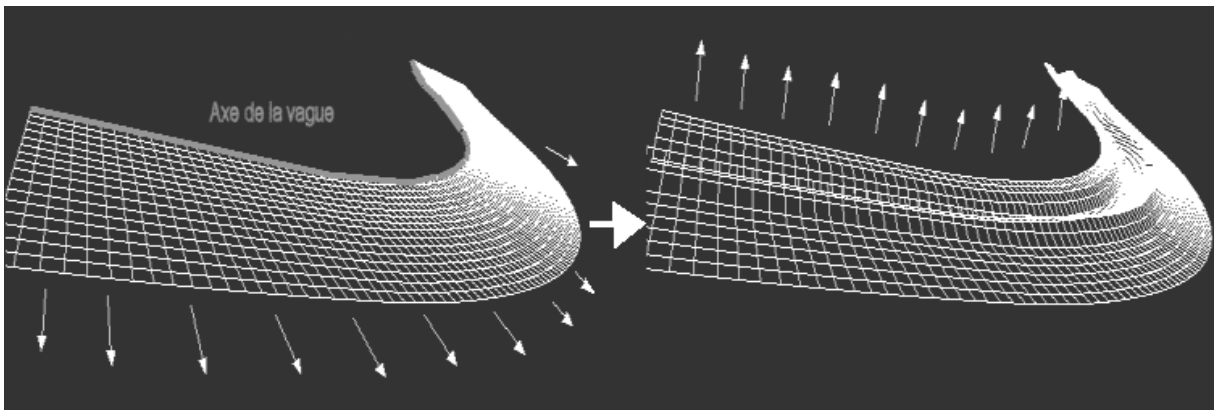


FIG. 3.4 – Création de la géométrie par épaississement le long des normales puis extrusion

P est le point de la vague, O l'origine du repère du quadrilatère, $(\vec{T},\vec{B},\vec{N})$ son repère local, et la fonction profil est $x \rightarrow f(x)$ sur [0..1] avec $x \in 0..L$ (figure 3.5).

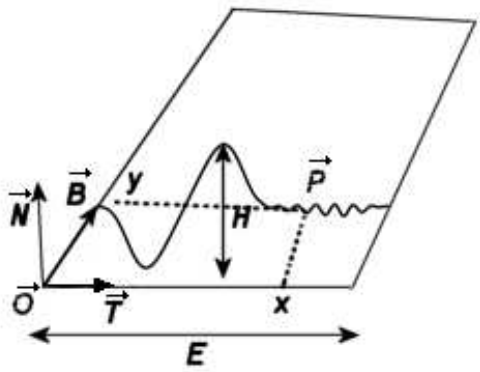


FIG. 3.5 – Génération d'une vague 3D à partir du profil 2D

Comme nous pouvons le voir sur la figure 3.6, chaque portion de vague possède un repère local orthonormal en plus du repère servant à générer le quadrilatère lui-même, ce dernier n'étant pas orthogonal (à cause du lissage de T). On peut facilement faire des transformations dans le repère orthonormal $(\vec{T},\vec{N},\vec{B})$, qui va nous servir à convertir les normales depuis le repère du profil 2D (N_p pré-calculées) vers le repère monde (N_w).

$\vec{N_w} = M\vec{N_p}$ *avec* $M = \begin{vmatrix} \vec{T} \\ \vec{N} \\ \vec{B} \end{vmatrix}$ et Np[x] précalulé (pour mémoire, $\vec{N_p} = \begin{vmatrix} \frac{f(x-1)-f(x+1)}{2/S} \\ 1 \\ 0 \end{vmatrix}$)

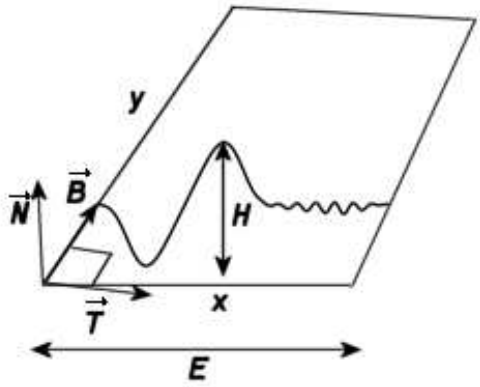


FIG. 3.6 – Transformation des normales en 3D à partir du profil 2D (notez que le repère est différent par rapport à la figure précédente)

Mise à l'échelle des normales

La géométrie que nous générons est calculée à partir d'un profil d'onde normalisé, nous effectuons une mise à l'échelle de ce profil pour créer notre géométrie. Les normales doivent donc également subir cette mise à l'échelle, ce qui nous évite de les recalculer complètement à la volée lors de la création du maillage (qui peut représenter jusqu'à plusieurs centaines de milliers de normales, et surtout qu'une différence finie à faible résolution serait moins bonne). Étant donnée un point P du profil 2D $f(x)$, $\vec{N}$ sa normale, E le facteur de mise à l'échelle en x (qui correspond à l'épaisseur de la vague) et H le facteur de mise à l'échelle en y (qui correspond à la hauteur de la vague), P' et $\vec{N'}$ le point et la normale mise à l'échelle :

$$\vec{P} = \begin{vmatrix} x \\ f(x) \end{vmatrix} \quad \vec{N} = \begin{vmatrix} \frac{\partial_y}{\partial_x} \\ 1 \end{vmatrix}$$

$$\vec{P'} = \begin{vmatrix} Ex \\ Hy \end{vmatrix} \quad \vec{N'} = \begin{vmatrix} \frac{H\partial_y}{E\partial_x} \\ 1 \end{vmatrix} = \begin{vmatrix} \frac{HN_px[x]}{EN_py[x]} \\ 1 \end{vmatrix}$$

La normales calculées par cette formule devront être renormalisées avant d'être utilisées pour des calculs d'éclairage.

3.4 Approche à base de bump mapping

L'autre représentation choisie concernant la génération de vagues est le rendu en bump mapping, permettant d'utiliser trés peu de géomtérie. Étant donné les opérations que nous effectuons lors de la construction du maillage 3D, nous pouvons faire mieux que du bump mapping 2D classique plaqué sur l'ensemble de la surface : nous pouvons à nouveau épouser la forme des vagues, en utilisant le profil comme une carte de bump 1D que nous allons utiliser le long de d'axe de la vague pour calculer à la volée les normales 3D. Cela va nous permettre une bien meilleure qualité visuelle pour beaucoup

moins de mémoire. Bien sûr, le bump ne permet plus de rendre compte de l'occultation visuelle des vagues, et introduit une petite distortion de parallaxe.

Les calculs à effectuer pour obtenir ce type de bump mapping un peu spécial sont exactement les mêmes que pour la génération du maillage, les seuls changements étant que tous les calculs sur les normales seront effectués coté GPU (dans le pixel shader) et qu'il faut générer des coordonnées de texture pour adresser la normal map 1D. Les coordonnées de texture sont générées très simplement : 0 du coté de l'axe de la vague, et 1 à l'autre bout du quadrilatère (figure 3.7).

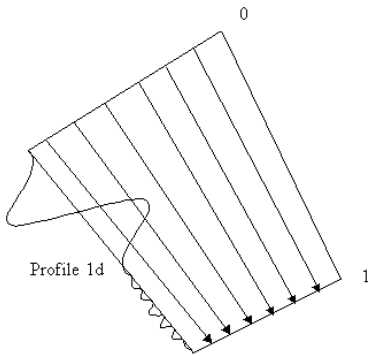


FIG. 3.7 – Un quadrilatère de la vague en bump mapping 1D

Un autre détail important concerne la normal map elle même : en général, les normal maps sont représentées sous la forme d'images dont les couleurs RGB codent les 3 composantes de la normale correspondante sous forme compacte, avec 8, 16 ou 32 bits par composante de couleur/normale. Ce compactage d'une valeur flottante dans un entier entraîne une perte de précision dépendante du nombre de bits choisis pour encoder la texture, qui peut être visible si de faibles variations de normales apparaissent dans la normal map sur une grande zone de la vague. Pour pallier ce problème, il est dorénavant possible (grâce au hard graphique récent) de stocker les normales dans une texture dont les composantes sont des flottants encodés sur 16 ou 32 bits, sans perte de performance visible pour l'utilisateur, ce qui permet de régler efficacement le problème.

Le hardware va s'occuper automatiquement de filtrer la normal map linéairement, mais des problèmes d'aliasing risquent quand même d'apparaître lorsque l'observateur va s'éloigner de l'onde, puisqu'alors la zone réelle représentée par un pixel à l'écran est très grande, ce qui revient à faire à un sous-échantillonnage. On observe alors des discontinuités dans la représentation de la scène, qui sont habituellement corrigées grâce au MIP-mapping. Le MIP-mapping fonctionne bien lorsqu'il s'agit de filtrer une texture standard, mais ici malheureusement il s'agit d'une carte de normales. On ne peut absolument pas faire la moyenne des normales pour générer la mip-map de niveau inférieur comme on le ferait avec une texture, puisqu'on aurait alors des reflets correspondants à des normales totalement nouvelles, toutes lisses (on perd souvent les reflets spéculaires dans ce cas). Il faut en fait essayer d'estimer les variations des normales statistiquement puis modifier l'environement mapping (comme dans [Tok04]), ce qui implique un calcul à la volée : le calcul des mip-maps devient donc un shader à part entière. Dans le cas de l'environment mapping, quand un rayon est réfléchit par une surface rendue en bump mapping, l'écart type des normales σ va être propagée par la réflection, ce

qui a pour effet de rendre floue l'image réfléchie. Cet effet peut être modélisé en calculant la mip-map de l'environment map comme une fonction de $\log_2 \sigma$.

3.5 Gestions des intersections de vagues

Lorsque deux vagues s'intersectent, les techniques présentées jusqu'ici ne suffisent plus : si l'on affiche les différentes vagues sans précaution, les effets optiques ne tiennent pas compte des deux vagues dans le croisement, et dans le cas du bump on observe en plus un phénomène de z-fighting : les polygones à l'intersection sont à la même position en z donc la carte graphique affiche tantôt les pixels de l'un, tantôt ceux de l'autre selon l'arrondi obtenu lors du calcul de projection.

Il est donc nécessaire de proposer des algorithmes spécifiques pour détecter puis traiter ces intersections de façon à ce qu'elles soient rendues conformément à la réalité, sans artefact visuel. Nous pouvons voir sur la figure 3.8 une intersection réelle d'onde de choc dans un ruisseau : pour avoir une intersection parfaite des deux vagues, il conviendrait de calculer l'intersection des bandelettes qui définissent les deux ondes, ce qui donne un quadrilatère à bords légèrement arrondis à générer spécifiquement. Pour conserver l'économie des profils 1D (et en particulier, générer un bump 2D spécifique), nous allons approximer les intersections par des quadrilatères, ce qui va nous permettre de nous positionner dans une configuration beaucoup plus simple pour le rendu en nous ramenant à un produit tensoriel de profils 1D et donc accélérer les calculs. Cependant, nous allons voir que cette simplification n'est pas sans poser certains problèmes lors du rendu.

FIG. 3.8 – Intersection d'ondes de choc dans un ruisseau réel

Les intersections entre les vagues peuvent être complexes : il peut y avoir des vagues qui s'intersectent plusieurs fois, et les croisements ont parfois une forme quelconque. Ici, nous ne nous sommes

intéressés qu'au cas de l'intersection standard "simple", où elle peut être assimilée à un quadrilatère. Pour détecter efficacement les intersections, nous testons dans un premier temps l'intersections des boîtes englobantes des vagues entre elles, pour éliminer rapidement les cas où aucune intersection n'est possible. Pour les couples de vagues où une intersection est possible, nous parcourons donc les vagues telles que nous les affichons, bande par bande, puis nous testons à nouveau les boites englobantes (alignées sur les axes, en 2D) de ces bandes. Si on détecte un recouvrement, nous effectuons un calcul d'intersection précis entre les deux quadrilatères pour trouver exactement l'intersection. Les test grossiers effectués en amont (sur les boites englobantes des vagues complètes) permettent d'éviter que les test d'intersections fassent écrouler les performances de l'application. De plus, nous ne testons effectivement que les vagues visibles à l'écran, ce qui permets parfois de diminuer les nombre de tests notamment lorsque l'on est proche du ruisseau.

3.5.1 Maillage 3D

Comme pour les ondes classiques, nous avons traité les façons différentes de représenter les croisements : avec de la géométrie ou en bump mapping, pour les mêmes raisons que celles évoquées plus haut. La génération des points pour le maillage 3D composant l'intersection se fera par simple interpolation linéaire en ce qui concerne les positions dans le plan d'eau, avec une possibilité de sous-échantillonnage comme dans le cas des ondes. D'un point de vue spectral, lorsque deux ondes se superposent, on additionne leur intensité, nous avons donc combiné les profils en prenant la somme des hauteurs. La figure 3.9 illustre la façon dont sont générés les sommets du maillage 3D pour les intersections.

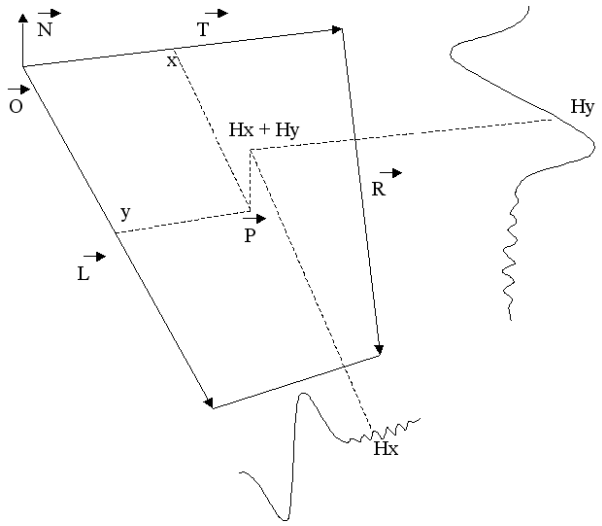


FIG. 3.9 – Génération du maillage pour l'intersection de 2 vagues

$\vec{O}$ étant l'origine du quadrilatère, $\vec{N}$ la normale au plan d'eau, x et y les coordonnées des points servant pour l'interpolation dans le quadrilatère ($x \in [0..1]$ et $y \in [0..1]$) :

$$\vec{P} = \vec{O} + x\vec{T} + (1-x)y\vec{L} + xy\vec{R} + (Hx+Hy)\vec{N}$$

Pour éviter de compliquer le tracé des vagues, qui ici est inchangé, le quadrilatère est dessiné par dessus, ce qui pose des problèmes d'occultation similaires au tracé du plan d'eau, que nous traitons maintenant.

Résolution des problèmes d'occultation

La génération de cette géométrie avec les simplifications apportées au modèle posent quelques problèmes : on observe à nouveau des problèmes d'occultation. Comme dans le cas du plan d'eau et des vagues, nous allons faire appel au stencil buffer pour effectuer des tests de recouvrement au pixel prés. Nous ne devons rien dessiner à l'endroit de l'intersection à l'exception de la géométrie générée pour cet effet, mais nous ne pouvons pas compter sur le z-buffer pour résoudre correctement la visibilité puisque $Hx+Hy$ peut être plus petit que Hx ou que Hy (si l'autre est négatif), ce qui laisserait apparaître la vague d'origine. On peut naïvement penser que la technique utilisée pour le plan d'eau et les vagues fonctionnera, mais cette fois il y a quatre primitive en compétition par endroits. Nous avons dû mettre au point une technique plus complexe qui permet de gérer correctement ces problèmes d'occultation, illustrée sur la figure 3.10. La géométrie d'intersection est dessinée en premier, elle positionne le stencil à 1 (bit 0 positionné) pour interdire tout recouvrement aux pixels tracés, puis on utilise ensuite le quadrilatère d'intersection de marquage (qui n'est dessiné que dans le stencil buffer) sous la forme de quatre triangles, qui positionnent le bit 1 (+2 dans le stencil) pour les triangles en vert clair, et le bit 2 (+4) pour les triangles en rose pâle, afin de préparer la sélection de quelle vague à le droit d'apparaître dans quelle zone. La vague qui passe du coté des triangles vert clair ne pourra être dessinée que pour les zone n'ayant pas les bits 0 et 1 de positionnés, celle qui passe de l'autre coté ne sera dessinée qu'aux endroits où les bits 0 et 2 ne sont pas positionnés. Cette technique nous assure un rendu correct tant que la géométrie de l'intersection ne déborde pas dans les autres triangles de stencil (verts et roses), ce qui peut se produire si la hauteur des vagues est trop forte où avec une vue très rasante. Dans ce cas de figure, il faudrait générer une géométrie (figure 3.11), qui colle parfaitement à l'affichage de l'intersection.

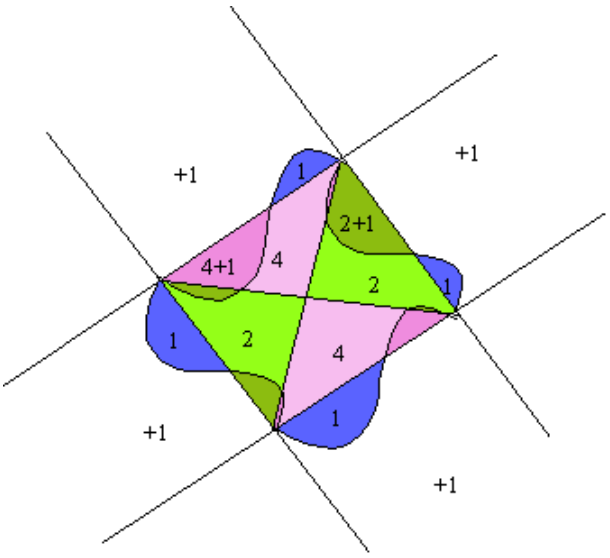


FIG. 3.10 – Algorithme de recouvrement pour les intersections (valeurs du stencil)

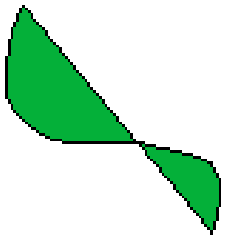


FIG. 3.11 – Géométrie spécifique pour les intersections

3.5.2 Bump mapping et calcul des normales

Les intersections de vagues peuvent également être représentées en bump mapping. L'intersection sera alors représentée par un unique quadrilatère sur lequel on appliquera un bump mapping 2D, combinaison des deux profils 1D qui ont servi à la génération des vagues. Le calcul sera exactement le même pour la version polygonale de l'intersection, la seule différence sera au niveau de l'implémentation qui sera portée sur GPU. Il s'agit donc d'effectuer un calcul adapté pour générer les normales à partir des deux profils 1D additionnés qui génèrent le quadrilatère d'intersection. Pour calculer ces normales, on ne peut pas juste ajouter les normales pré-calculées des deux profils 1D et normaliser le tout, car les normales ne sont pas additives (on l'a deja vu pour le MIP-mapping). Si l'on considère les surfaces S_1 et S_2 des deux vagues à additionner au croisement, on peut écrire les normales $\vec{N_1}$ et $\vec{N_2}$ sous la forme :

$$N_1 = \frac{\vec{dS_1}}{\left\|\vec{dS_1}\right\|} \quad et \quad N_2 = \frac{\vec{dS_2}}{\left\|\vec{dS_2}\right\|}$$

où $\vec{dS}$ est obtenu par produit vectoriel ($\vec{dS} = \begin{vmatrix} -\partial h/\partial x \\ -\partial h/\partial y \\ 1 \end{vmatrix}$ pour h(x,y) dans un espace normalisé).

Il se trouve que cette représentation est, elle, additive, nous pouvons calculer la somme $\vec{dS_\Sigma}$:

$$\vec{dS_\Sigma} = \begin{vmatrix} dS_1x \\ dS_1y \\ 1 \end{vmatrix} + \begin{vmatrix} dS_2x \\ dS_2y \\ 1 \end{vmatrix} = \vec{dS_1} + \vec{dS_2} - \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}$$

Or nous avons $dS_1 = \frac{1}{N_1z}\vec{N_1}$ et $dS_2 = \frac{1}{N_2z}\vec{N_2}$, ce qui nous amène à :

$$\vec{N} = \frac{1}{N_1z}\vec{N1} + \frac{1}{N_2z}\vec{N2} - \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}$$

Nous utiliserons donc cette formule pour calculer les normales des intersections à partir des deux profils combinés. Nous allons pouvoir utiliser les cartes de normales 1D pré-calculées qui contiennent les normales $\vec{N_1}$ et $\vec{N_2}$ pour effectuer ce calcul à la volée en quelques opérations seulement, ce qui permet une implémentation efficace dans un shader pour le bump mapping, mais également sur CPU pour la version géométrique.

Chapitre 4

Resultats obtenus

4.1 Reflexions, réfraction et autres effets optiques

Pour obtenir un ruisseau réaliste, nous avons implémenté les différents effets dont nous avons parlé précédemment, à savoir un shader qui fait de la réflection et de la réfraction, le tout contrôlé par le coefficient de Fresnel. La réflection à été implémenté grâce à une cube map d'environnement, dont nous nous servons pour afficher le décor lointain qui entoure le ruisseau. Voici une image du ruisseau en mode "Miroir", où l'on aperçoit uniquement les réflections :

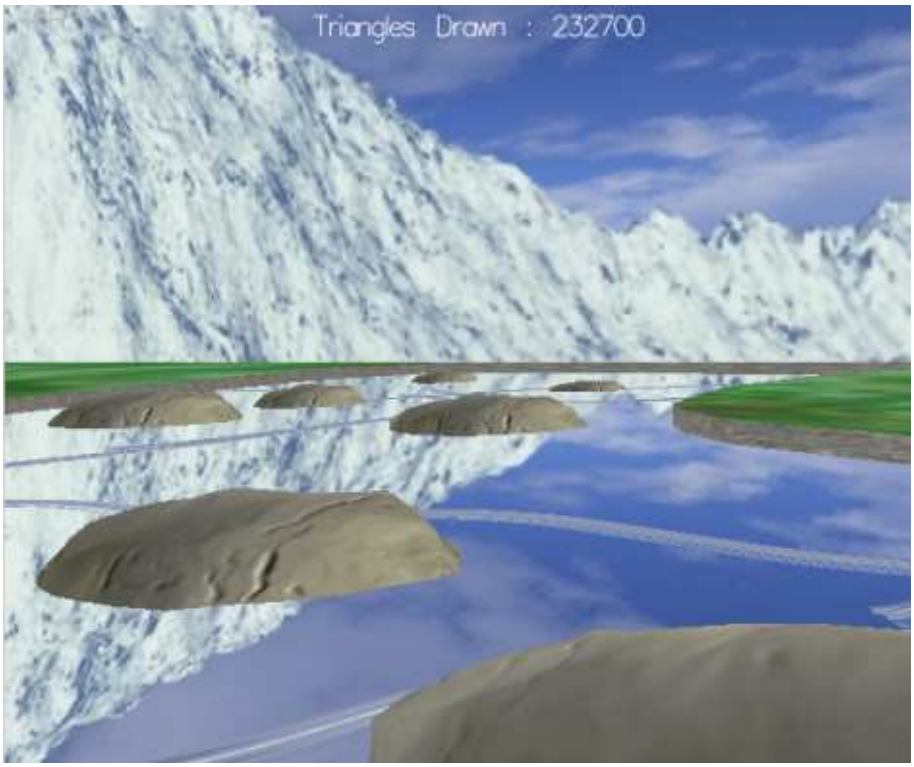


FIG. 4.1 – Le ruisseau en mode "Miroir"

La réfraction à été implémentée grâce au système de textures projectives dont nous avons parlé dans la section 2.3.5. On effectue un rendu en deux passes : durant la première, on rend dans une

texture tout ce qui est au-dessous du niveau de l'eau (tout ce qui peut être réfracté). On peut s'assurer qu'aucune géométrie au-dessus du plan d'eau n'est rendue en utilisant un plan de clipping utilisateur situé au même niveau que le plan d'eau, c'est pour cela que le rocher est coupé en deux et l'on ne voit que sa partie immergée. Durant la deuxième passe, celle de rendu final, on décale le point à réfracter selon la direction de réfraction, puis on projette ce point dans l'espace de la caméra pour enfin adresser la texture de réfraction, avec une mise à l'échelle appropriée. Sur la figure 4.2, on peut voir qui réfracte le fond du ruisseau et la texture de réfraction, en bas à gauche de l'image.

FIG. 4.2 – Une vague qui réfracte le fond du ruisseau.

Nous avons codé en tout 3 vertex shaders et 3 pixel shaders. La première paire sert au rendu du plan d'eau, avec un calcul d'illumination par pixel et des optimisations spécifiques dues à sa normale constante. La seconde sert à rendre la géométrie des vagues, ici le calcul d'illumination est effectué à chaque sommet, puisque les triangles représentant les vagues sont relativement petits (quelques pixels à l'écran), il n'était pas nécessaire d'effectuer un calcul par pixel. Et enfin un shader pour le bump mapping, qui effectue les calculs d'illumination par pixel.

Pour obtenir un rendu plus réaliste, nous avons pris en compte l'atténuation visuelle que provoque la quantité d'eau à travers laquelle on regarde. En effet, plus la masse d'eau à travers laquelle on regarde est profonde, moins on aperçoit ce qui est au fond. En clair, lorsque l'on regarde le ruisseau avec un angle rasant, on perçoit bien mieux les obstacles immergés qui sont proche que ceux qui sont à l'autre bout du ruisseau, puisque la masse d'eau qui nous sépare des obstacles lointains est bien plus épaisse que celle qui nous sépare des obstacles proches.

Voici ce que l'on peut obtenir en termes visuel avec notre application pour un ruisseau complet :

FIG. 4.3 – Le ruisseau rendu par notre application

Sur les figure 4.4 et 4.5, on peut voir une vague en gros plan. On peut apercevoir distinctement les ondes de capillarité, et les différents effets optique implémentés.

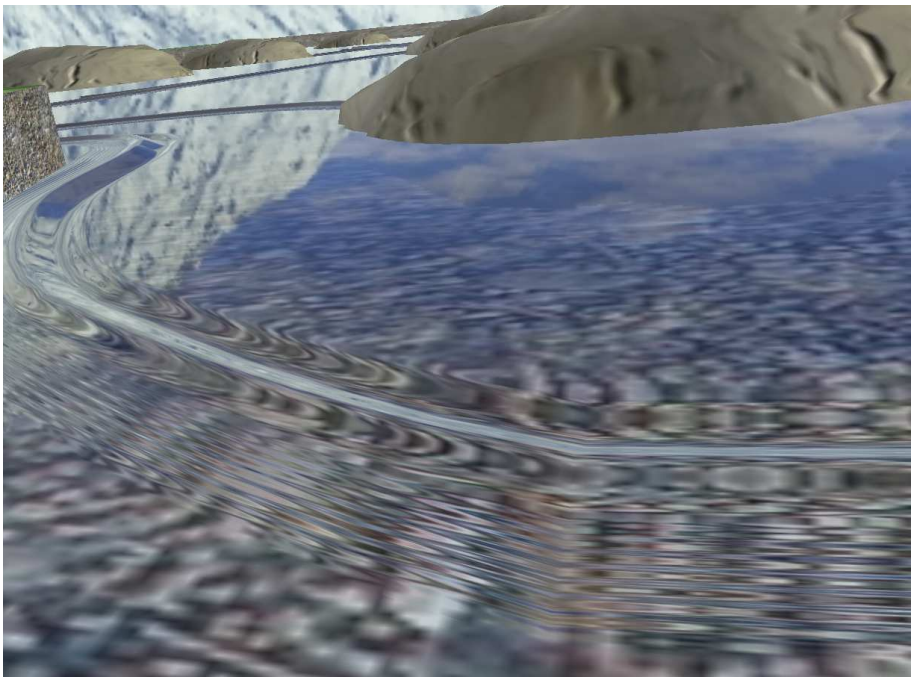


FIG. 4.4 – Une vague en gros plan

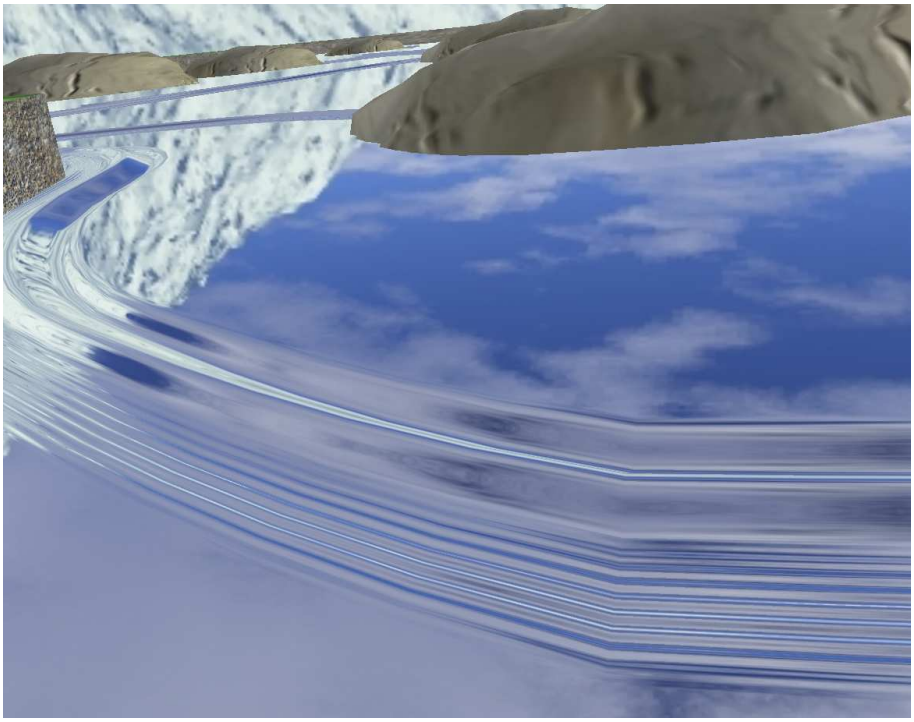


FIG. 4.5 – La même vague en mirroir

4.2 Performance

4.2.1 Niveaux de détails

Nous allons ici détailler l'impact des différentes méthodes de rendu sur les performances, sur un exemple complet de ruisseau assez complexe.
Pour obtenir le graphique 4.6, nous avons utilisé deux points de vue différents : un point de vue lointain d'où on peut apercevoir la totalité du ruisseau, et un autre plus rapproché d'où on ne voit pas toutes les ondes. Nous avons passé en revue tous les niveaux de détails possible qu'offre notre implémentation : les différents niveaux de sous-échantillonnage du maillage 3D, le bump mapping, et enfin une méthode automatique qui détermine automatiquement le niveau de détail/modèle à choisir en fonction du point vue (bump ou maillage 3D, avec un sous-échantillonnage de 1 à 64 le long du profile et de 1 à 4 le long de l'axe de la vague). Les tests suivants on été effectués sur un Athlon 1.2Ghz, avec une GeForce 6600 en AGP 4x (la performance du bus AGP a un impact assez important ici puisque nous envoyons souvent des triangles à la carte graphique, il peut donc éventuellement faire office de goulot d'étranglement).

Voici les résultats obtenus :

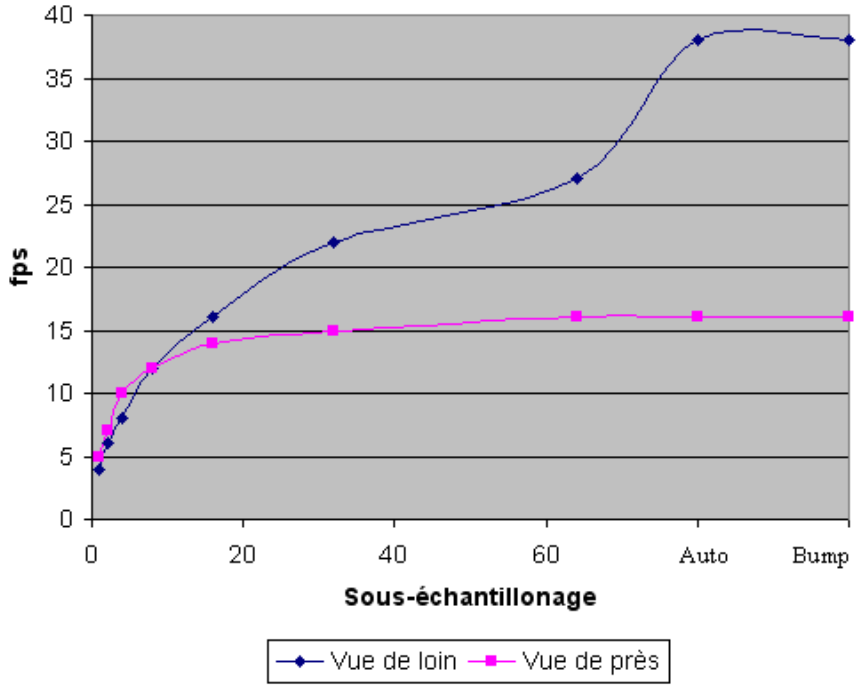


FIG. 4.6 – Les performances de l'affichage en fonction du niveau de détail utilisé

Nous pouvons voir que le bump mapping est le plus performant dans les deux cas, à égalité avec notre méthode de sélection automatique de niveaux de détails, sauf que cette dernière choisit le meilleur niveau de détail au en termes de la qualité visuelle.

On peut également remarquer que sur la vue de prés, un sous-échantillonnage supérieur à 16 n'affecte pas les performances, et donc à partir de ce moment ce n'est plus le nombre de triangles générés qui représentent le goulot d'étranglement de l'application. Nous avons donc cherché quel était ce nouveau facteur de ralentissement : il existe un certain nombre de tests à faire passer à l'application pour trouver quel partie du pipeline graphique est responsable du ralentissement [RB04]. Les calculs d'illumination lors de l'affichage des vagues sont effectués par sommet, or nous avons mesuré que l'application n'était pas sensible au changement du nombre de sommets, et le pixel shader utilisé pour effectuer le rendu des vagues est très simple (deux accès texture et une interpolation linéaire). Après avoir passé au crible tous les tests mentionnés dans [RB04], nous en arrivons à la conclusion que notre application est limitée par le CPU, ce qui à été confirmé par un profiling : on passe cinq fois plus de temps dans la fonction qui calcule la géométrie que dans sa fonction de rendu, et on passe aussi énormément de temps à détecter les intersections (10 fois plus de temps qu'au rendu des vagues), sans compter le temps que prend la simulation elle-même (1,3 fois plus que le temps de rendu total d'une image), ce qui nous montre que de nombreuses améliorations pourraient encore être développées en ce qui concerne les performances de l'application.

4.2.2 Répartition des temps de calcul

Ces analyses nous ont mené a mesurer plus en détail le temps pris par les différentes taches qu'effectue l'application durant la construction d'une image. Nous avons identifié 4 taches : la simulation

elle-même (CPU), la construction et le calcul de la géométrie (CPU), le rendu lui-même (GPU) et enfin les calculs d'intersections (CPU). Les mesures données par le tableau 4.1 ont été effectuées sur un survol de ruisseau (le même qu'au dessus) d'environ 13 secondes, avec des passages en plan large, des zooms, le tout en mode de section de détail automatique (qui doit théoriquement nous donner toujours la meilleure représentation possible).

Simulation	2426 ms
Géométrie	2026 ms
Intersections	1090 ms
Rendu	463 ms

TAB. 4.1 – Mesures de performance des 4 taches principales

Notez que les temps indiqués sont le cumul des temps pris par la tâche en question sur les 13 secondes de survol. Le temps restant est pris en majeure partie par le chargement de l'application, des shaders et le calcul de $\nabla\phi = 0$ au début du programme. Il convient plutôt de comparer les 4 valeurs relativement : la simulation prend à peu prés autant de temps que la création de la géométrie et de son rendu, mais nous constatons que la génération de triangles engouffre 2.5 fois plus de temps que le rendu des vagues lui-même, bien que nous ayons utilisé ici une sélection automatique du niveau de détail. Deux explications sont possibles : soit notre sélection automatique génère plus de détails qu'il n'est nécessaire (mais c'est difficile à mesurer), soit les calculs de géométrie sont lents, soit un peu des deux. On peut envisager d'optimiser ces calculs avec un traitement en parallèle du calcul des vagues (avec les instructions SIMD spécifiques aux Pentium IV ou aux Athlons Xp), mais il serait plus judicieux d'essayer de régler au mieux le sous-échantillonnage pour avoir un résultat visuel et une performance optimale, avec un méthode moins empirique que celle que nous avons employée.
Nous pouvons également nous apercevoir que les calculs d'intersections prennent beaucoup de temps, il serait donc judicieux de prévoir un calcul "hiérarchique" des intersections : on pourrait tester les collisions grossièrement avec une grille puis ensuite raffiner les tests si on constate que les boites englobantes des bandes se recouvrent, au lieu de parcourir directement toute la vague à chaque fois.

4.3 Discussion

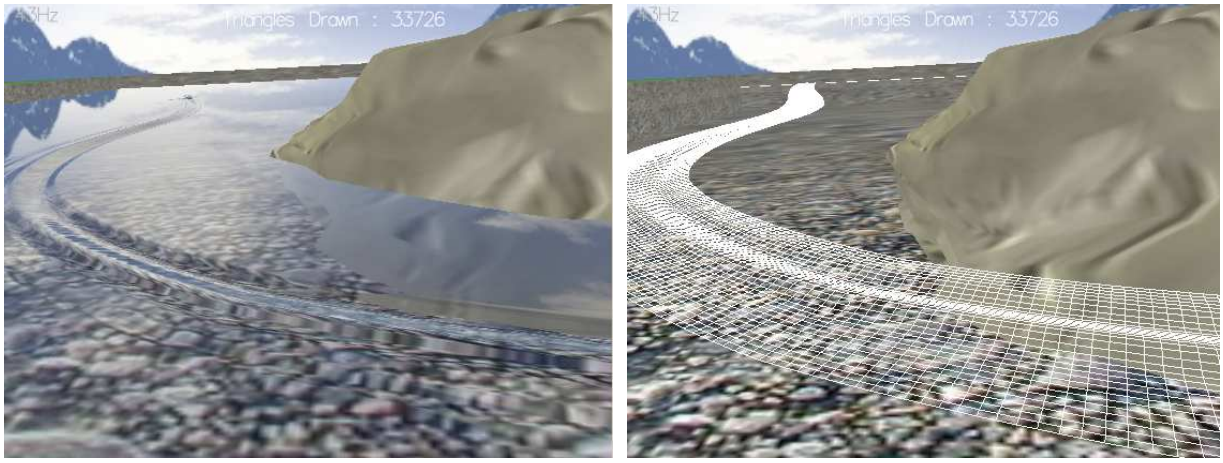
4.3.1 Niveaux de détails : qualité visuelle

Nous allons ici comparer les différents niveaux de détail utilisables sur une vague en termes de qualité visuelle. Dans un premier temps observons une vague générée en maillage 3D :

(a) (b)

FIG. 4.7 – Une vague avec beaucoup de détails (facteur de sous-échantillonnage : 8)

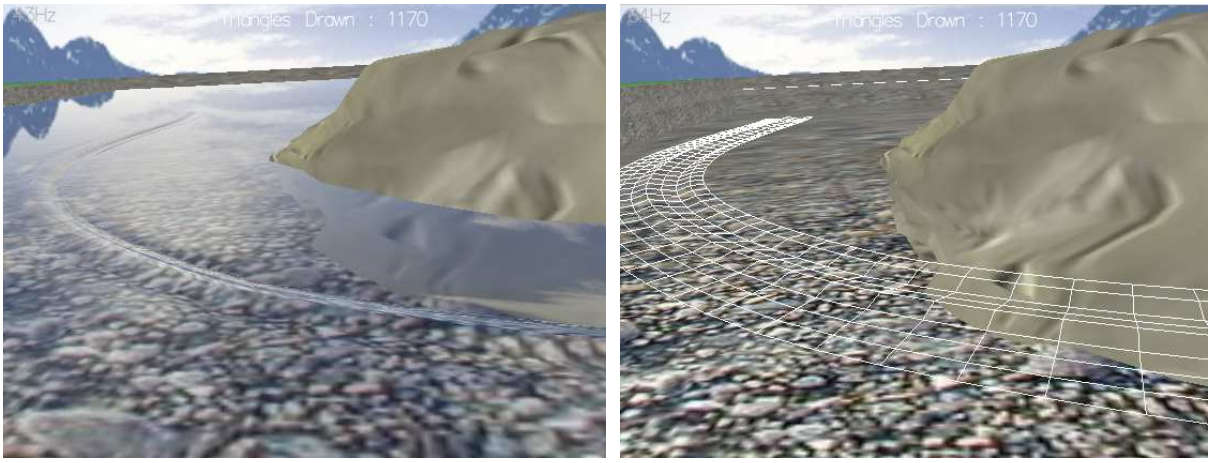
Cette vague comporte en tout 110000 triangles, on peut observer la bonne qualité de rendu. Notons les reflets et l'occultation corrects grâce à l'échantillonnage élevé, mais surtout les ondes capillaires en amont de la vague. On peut se rendre compte de la taille des quadrilatères à l'écran, ils sont très petits (quelques pixels).



(a) (b)

FIG. 4.8 – La même vague moyennement détaillée (facteur de sous-échantillonnage : 16)

Dans cette version nous n'avons plus que 33000 triangles, mais on peut difficilement distinguer la différence avec la version d'au-dessus. Seules les ondes capillaires sont affectées (on pourrait les traiter spécifiquement en bump). On peut donc dire que la version précédente était trop échantillonnée, et on en tiendra compte lors du choix d'un niveau de détail automatique : il est inutile de sur-échantillonner les vagues, d'autant que l'on peut voir qu'on a gagné 18 image par secondes (de 25 à 43 hz) avec cette version de la vague. Les triangles ont une taille plus optimale sans toutefois devenir un problème pour que l'on distingue correctement la géométrie de la vague.

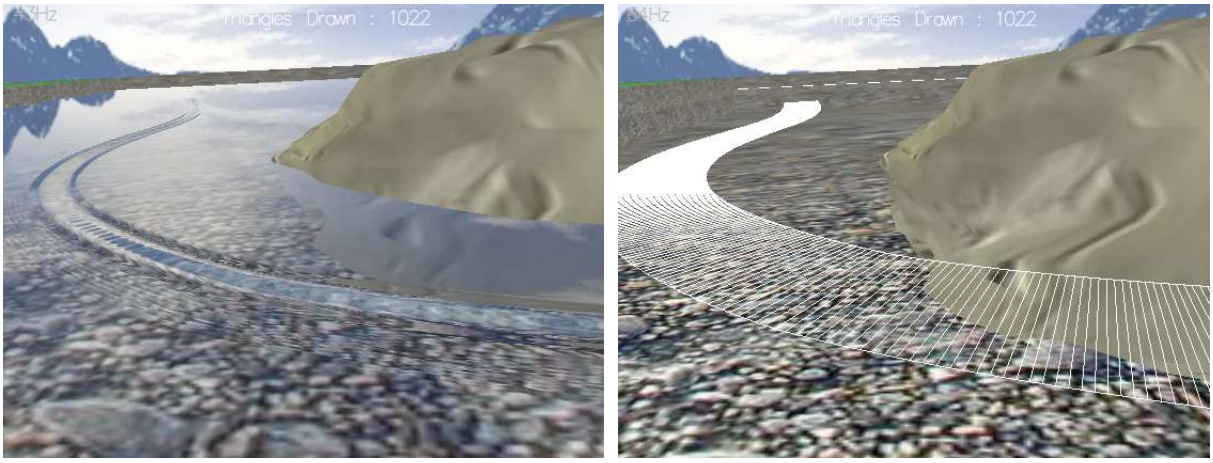


(a) (b)

FIG. 4.9 – La même vague avec peu de détails (facteur de sous-échantillonnage : 64)

Avec la version la moins détaillée possible il ne reste que 1000 triangles, mais on voit clairement apparaître des problèmes : l'occultation est mauvaise, les reflets et les distortions s'estompent et on a l'impression que la vague s'est écrasée : on peut comprendre pourquoi en regardant la version en fil de fer, le sous-échantillonnage ne permet pas de représenter correctement la vague surtout dans les endroits où la pente du profile est élevée, ce qui génère des normales trop approximées qui faussent les calculs optiques.

Observons maintenant la même onde rendue en bump mapping :



(a) (b)

FIG. 4.10 – La même vague en bump mapping

On peut voir que le rendu est plutôt satisfaisant, si l'on considère que l'on a à peu prés le même nombre de triangles que dans la version la moins détaillée de la vague. On remarque toutefois que l'occultation n'est pas bonne et que la parallaxe n'est pas correcte. On réservera donc le bump aux vues à angle non rasant, et on pourra grâce à une technique de parrallax mapping ([Wel04]), corriger le manque de parallaxe.

4.3.2 Aliasing

Comme nous l'avons évoqué précédemment, un pixel de l'image contient l'information d'une zone de la scène dépendante de la distance et de l'angle entre la surface et l'observateur. Plus la distance est grande plus la zone est importante. De même, plus l'angle est rasant, plus la zone est grande. Lorsqu'un pixel corresponds à une grande zone de la scène, avec un rendu standard on obtiens de l'aliasing. On peut observer ce genre de phénomène aussi bien avec de la géométrie (figure 4.11 qu'avec le bump mapping. Pour éviter ce genre de problème dans le cadre d'un rendu classique, on peut faire au MIP-mapping sur les textures d'environnement, mais dans notre cas c'est insuffisant (les textures de la figure 4.11 sont MIP-mappées), puisque le problème principal provient du sur-échantillonnage des normales dans un même pixel. L'aliasing est surtout visible sur les ondes de capillarité :

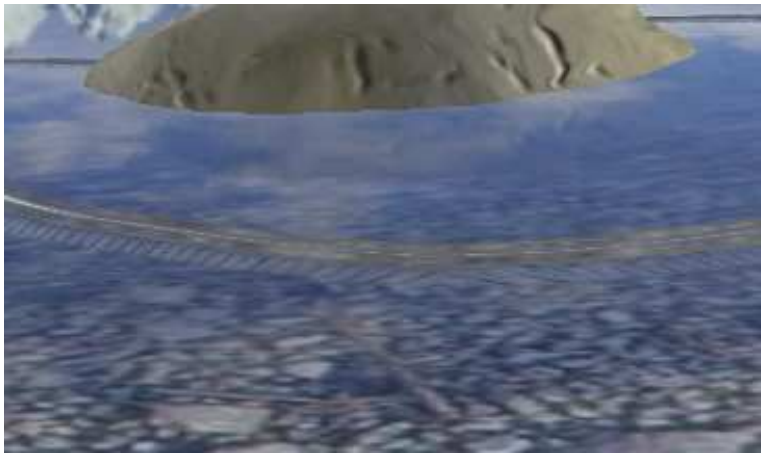


FIG. 4.11 – Aliasing géométrique sur les ondes de capillarité

On peut alors penser qu'il suffit de sous-échantillonner la géométrie pour que le phénomène disparaisse, ce qui est effectivement le cas (figure 4.12), mais le sous-échantillonnage estompe les ondes de capillarité comme nous l'avons vu dans la section 4.3.1. De plus la sélection automatique du bon niveau de sous-échantillonnage est loin d'être évidente, elle pourrait constituer un sujet d'étude à part entière. L'idéal serait de mettre en place un shader spécifique simulant un matériau dont la rugosité représente les détails [Kaj85], qui tienne compte de la zone représentée dans son ensemble, mais malheureusement ce genre de calcul peut s'avérer coûteux, et alourdir le rendu.

FIG. 4.12 – En sous-échantillonnant, l'aliasing disparaît

4.3.3 Intersections de vagues

Les croisements d'ondes tels que nous les avons implémentés donnent une forme complexe caractéristique dont le rendu est plutôt satisfaisant. En voici quatre vues de loin :

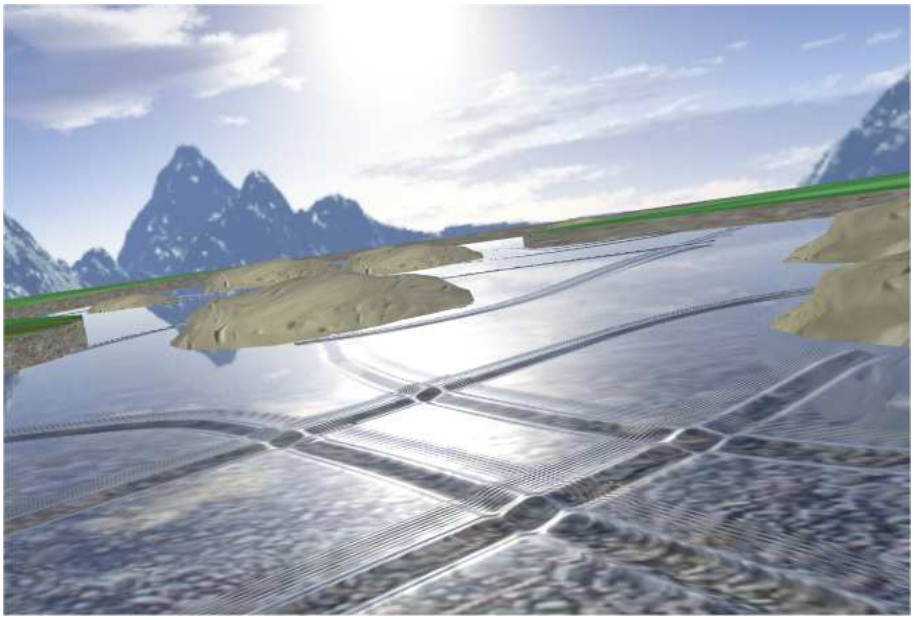


FIG. 4.13 – quatre intersections.

En voici une en gros plan :

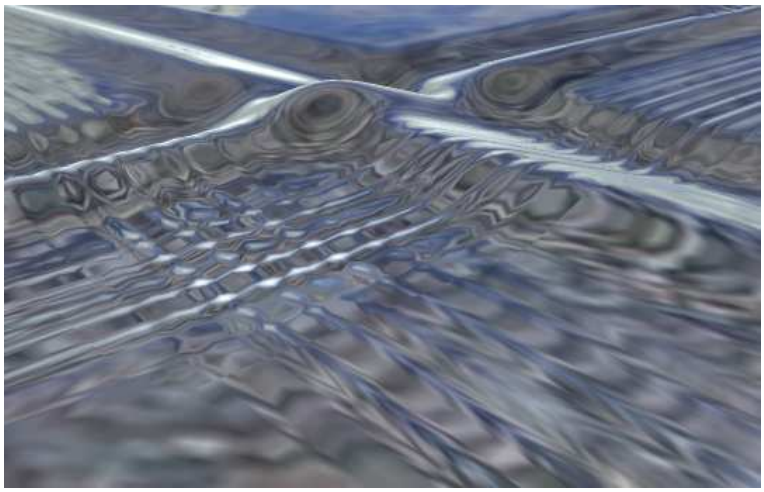


FIG. 4.14 – Une intersection de vagues, calculée par notre application

Il y a néanmoins quelques artefacts notamment au niveau des raccords avec les deux autres vagues : l'approximation de l'intersection par un quadrilatère laisse parfois un trou entre l'intersection et la vague. De plus, notre méthode de d'affichage avec le stencil buffer pose parfois quelques problèmes : quand plus de deux vagues se croisent presque au même endroit, le fait d'afficher plusieurs couches de géométrie empêche notre algorithme de fonctionner correctement (figure 4.15), et il est très sensible à l'ordre d'affichage des vagues.

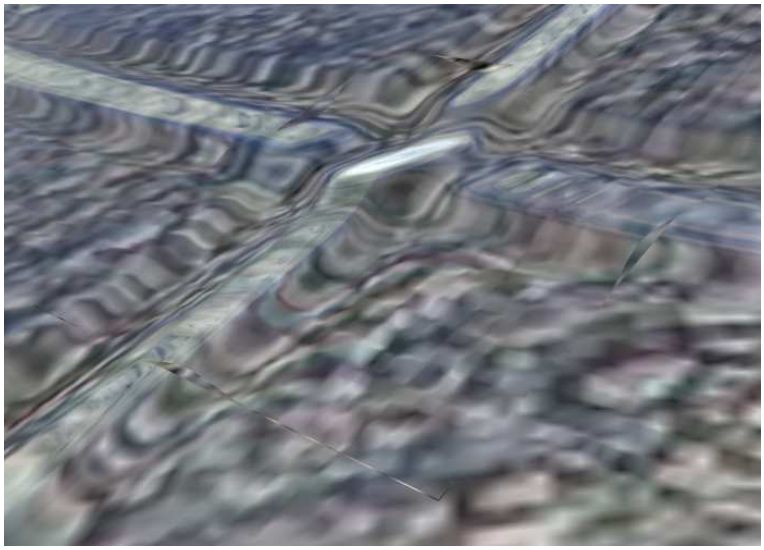


FIG. 4.15 – Un problème de raccord sur une intersection

De plus nous ne traitons que les intersection "propres", c'est à dire celles qui ont quatre cotés, mais on obtient assez souvent des croisements plus complexes (figure 4.16), et ceux-ci ne peuvent pas être facilement traitées avec une méthode similaire à la notre, cela demanderait des calculs trop complexes.

FIG. 4.16 – Une intersection complexe

Chapitre 5

Conclusion et perspectives

Nous avons proposé une méthode adaptative pour le rendu de ruisseaux en temps réel : l'approche phénoménologique de la simulation de ruisseaux nous a permis d'utiliser et de combiner différents modèles et techniques d'affichage afin d'obtenir un rendu interactif de ruisseau 3D en temps réel à haute qualité visuelle pour un faible taux de calcul et d'occupation mémoire. Les calculs sont concentrés uniquement sur la partie visible des phénomènes pertinents du ruisseau, allant des ondes de choc aux capillaires, difficilement représentables avec d'autres méthodes.

Les perspectives offertes par cette approche sont nombreuses : pour obtenir une animation et un rendu plus pertinents, il conviendrait de trouver un manière élégante de représenter les remous perturbateurs, qui donneraient un aspect beaucoup plus réaliste au ruisseau lorsqu'il est en mouvement, par exemple avec une géométrie adaptée. D'autres éléments pourraient également être modélisés pour obtenir plus de réalisme : l'écume sur les vagues, mais également un traitement plus poussé des interactions lumineuses de la lumière du soleil avec la surface de l'eau, en ajoutant des caustiques par exemple. Un traitement de l'aliasing plus poussé pourrait aussi contribuer grandement au réalisme de la scène. Le calcul des croisements de vagues est lui aussi perfectible : il faudrait obtenir de meilleurs raccords avec les vagues, et traiter les intersections complexes.

Étant donnés les résultats obtenus, nous espérons que nos travaux ont démontré l'intérêt d'une approche phénoménologique et de l'utilisation d'une hiérarchie de modèles multi-échelle telle que nous l'avons exposée pour le rendu de ruisseaux.

Bibliographie

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools : JGT*, 5(1) :9–22, 2000.

[AVO02] John Isidoro Alex Vlachos and Chris Oat. Rippling reflective and refractive water. In *ShaderX : Vertex and Pixel shader Programming Tips and Tricks*. Addison-Wesley, 2002.

[Bec92] Barry Glenn Becker. Smooth transitions between rendering algorithms during animation. Master's thesis, U. of Calif., Davis, 1992.

[Bel03] Vladimir Belyaev. Real-time simulation of water surface. In MAX Press, editor, *Conference Proceedings*, February 2003. http ://visinfo.zib.de/EVlib/Show/ ?EVL-2003-263.

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78 : Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 286–292, New York, NY, USA, 1978. ACM Press.

[BN76] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10) :542–547, 1976.

[CE01] David Cline and Parris K. Egbert. Terrain decimation through quadtree morphing. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(1), pages 62–69. IEEE Computer Society, 2001.

[DWS+97] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain : Real-time optimally adapting meshes. In *IEEE Visualization '97*, October 1997.

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02 : Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[EWWL98] J. P. Ewins, M. D. Waller, M. White, and P. F. Lister. MIP-map level selection for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 4(4) :317–329, October/December 1998.

[Fin04] Mark Finch. Effective water simulation from physical models. In *GPU Gems*. Addison-Wesley, 2004.

[FM98] Alexey V. Federov and W. Kendal Melville. Nonlinear gravity-capillary waves with forcing and dissipation. *J. Fluid Mech.*, 354 :1–42, 1998.

[FR86] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86 : Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1986. ACM Press.

[JR03] Kilgard Mark J. and Fernado Randima. *The Cg tutorial*. Addison-Wesley, 2003.

[Kaj85] James T. Kajiya. Anisotropic reflection models. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 15–21, July 1985.

[Lom04] Yann Lombard. Realistic natural effect rendering : Water 1, 2004.

[Lov03] Jörn Loviscach. Complex water effects at interactive frame rates. In V. Skala, editor, *Journal of WSCG*, volume 11, February 2003.

[NDW93] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide.* Addison-Wesley, 1993.

[NP01] Fabrice Neyret and Nathalie Praizelin. Phenomenological simulation of brooks. In *Computer Animation and Simulation*, pages 53–64. Eurographics, Springer, Sep 2001. Eurographics Workshop on Animation and Simulation, Manchester.

[Per85] Ken Perlin. An image synthesizer. *Computer Graphics*, 19(3) :287–296, July 1985.

[PN01] Ken Perlin and Fabrice Neyret. Flow noise. *Siggraph Technical Sketches and Applications*, page 187, Aug 2001.

[RB04] Ashu Rege and Clint Brewer. Practical performance analysis and tunning, 2004.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics*, 26(2) :249–252, July 1992.

[Sta99] Jos Stam. Stable fluids. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, Annual Conference Series, pages 121–128, Los Angeles, 1999. ACM Siggraph, Addison Wesley Longman.

[Sta01] Jos Stam. A simple fluid solver based on the fft. *J. Graph. Tools*, 6(2) :43–52, 2001.

[Tok04] Michael Toksvig. Mipmapping normal maps, 2004.

[Wel04] Terry Welsh. Parallax mapping with offset limiting : A per-pixel approximation of uneven surfaces, 2004.

[Wlo04] Matthias Wloka. Fresnel reflections, 2004.

[Zel04] Jeremy Zelsnack, 2004. Vertex Texture Fetch Water.